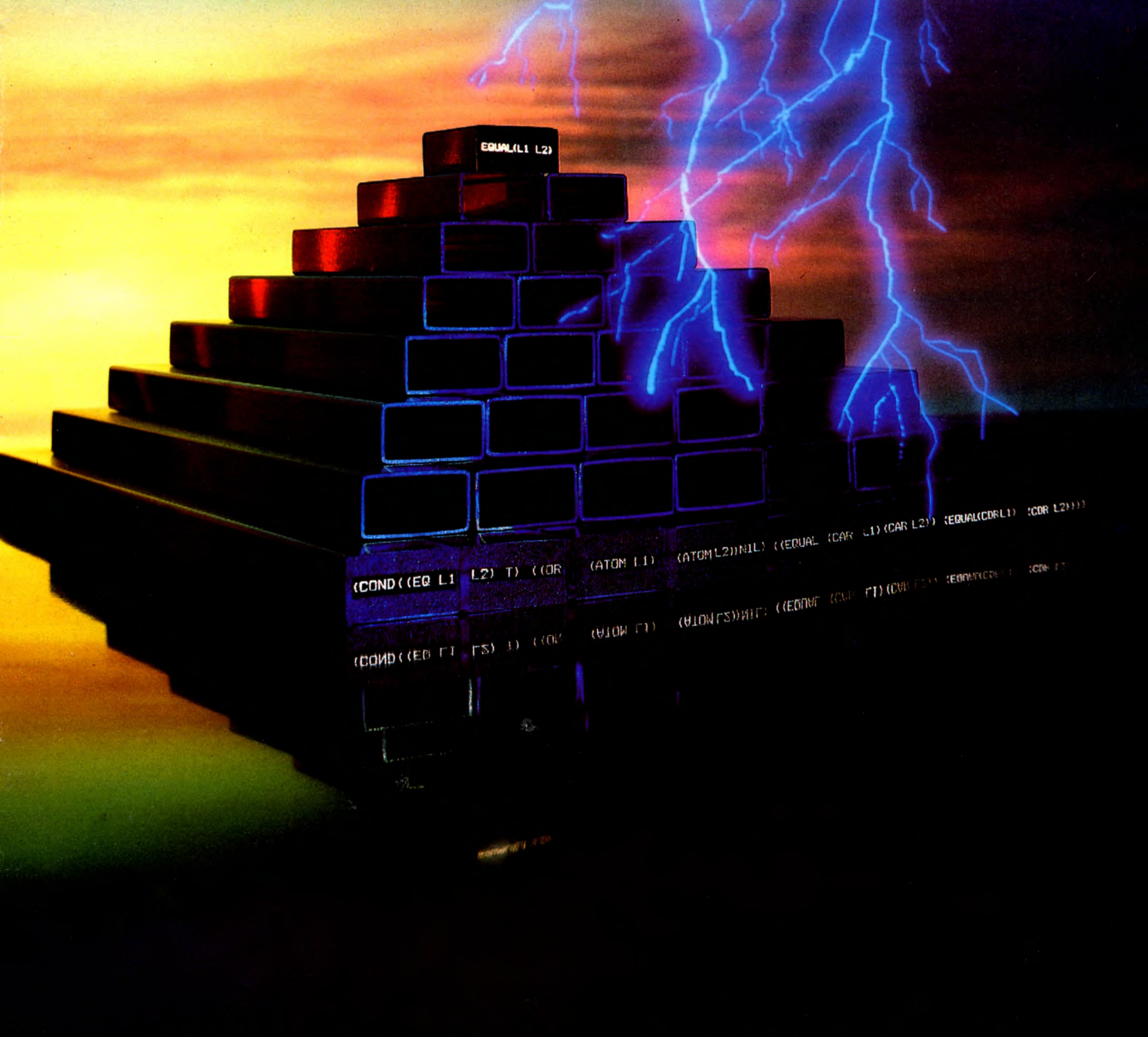
A MARSHALL CAVENDISH 46 COMPUTER COURSE IN WEEKLY PARTS

INPUT

LEARN PROGRAMMING – FOR FUN AND THE FUTURE



UK £1.00 Republic of Ireland £1.25 Malta 85c Australia $2.25 New Zealand $2.95

INPUT

Vol. 4 **No 46**

INDEX

The last part of INPUT, Part 52, will contain a complete, cross-referenced index. For easy access to your growing collection, a cumulative index to the contents of each issue is contained on the inside back cover.

PICTURE CREDITS

Front cover, Digital Arts. Pages 1429, 1430, 1432, Harry North. Page 1435, Lorea Carpenter. Pages 1436, 1439, Peter Reilly. Pages 1440, 1443, 1444, 1446, Paddy Mounter. Page 1449, Digital Arts. Pages 1450, 1452, 1454, 1455, Artist Partners/Stuart Robertson. Pages 1456, 1459, 1460, Graeme Harris.

Published by Marshall Cavendish Partworks Ltd, 58 Old Compton Street, London W1V 5PA, England. Typeset by MS Filmsetting Limited, Frome, Somerset. Printed by Cooper Clegg Web Offset Ltd, Gloucester and Howard Hunt Litho, London.

HOW TO ORDER YOUR BINDERS

UK and Republic of Ireland:
Send £4.95 (inc p & p) (IR£5.95) for each binder to the address below:
Marshall Cavendish Services Ltd, Department 980, Newtown Road, Hove, Sussex BN3 7DN

Australia: See inserts for details, or write to INPUT, Times Consultants, PO Box 213, Alexandria, NSW 2015

New Zealand: See inserts for details, or write to INPUT, Gordon and Gotch (NZ) Ltd, PO Box 1595, Wellington

Malta: Binders are available from local newsagents.

There are four binders each holding 13 issues.

BACK NUMBERS

Back numbers are supplied at the regular cover price (subject to availability).

UK and Republic of Ireland:
INPUT, Dept AN, Marshall Cavendish Services, Newtown Road, Hove BN3 7DN

Australia, New Zealand and Malta:
Back numbers are available through your local newsagent.

COPIES BY POST

Our Subscription Department can supply copies to any UK address regularly at £1.00 each. For example the cost of 26 issues is £26.00; for any other quantity simply multiply the number of issues required by £1.00. Send your order, with payment to:
Subscription Department, Marshall Cavendish Services Ltd, Newtown Road, Hove, Sussex BN3 7DN

Please state the title of the publication and the part from which you wish to start.

HOW TO PAY: Readers in UK and Republic of Ireland: All cheques or postal orders for binders, back numbers and copies by post should be made payable to:
Marshall Cavendish Partworks Ltd.

QUERIES: When writing in, please give the make and model of your computer, as well as the Part No., page and line where the program is rejected or where it does not work. We can only answer specific queries – and please do not telephone. Send your queries to INPUT Queries, Marshall Cavendish Partworks Ltd, 58 Old Compton Street, London W1V 5PA.

INPUT IS SPECIALLY DESIGNED FOR:

The SINCLAIR ZX SPECTRUM (16K, 48K, 128 and +), COMMODORE 64 and 128, ACORN ELECTRON, BBC B and B+, and the DRAGON 32 and 64.

In addition, many of the programs and explanations are also suitable for the SINCLAIR ZX81, COMMODORE VIC 20, and TANDY COLOUR COMPUTER in 32K with extended BASIC. Programs and text which are specifically for particular machines are indicated by the following symbols:

SPECTRUM 16K, 48K, 128, and +
COMMODORE 64 and 128
ACORN ELECTRON, BBC B and B+
DRAGON 32 and 64
ZX81
VIC 20
TANDY TRS80 COLOUR COMPUTER

PLANNING THE BEST COURSE

- PLANNING A PROJECT
- A CRITICAL PATH
- EFFICIENCY
- PERT CHARTS
- SAVING TIME

If you are trying to organise any sort of project—from servicing your car to buying a house—then this program can help you plan it out, and save you time as well

Have you ever found yourself half-way through a project such as decorating your living room and suddenly realised that if only you had planned the job properly you wouldn't now be sitting around waiting for the filler to dry before getting on with the painting? Or when you're fixing the car, found that if you'd planned it out properly you would have made sure that you replaced the stoplight switch before putting back the brake cylinder so that you now don't have to take it apart once again.

Any complex activity needs a certain amount of planning but sometimes there seem to be so many things you need to do, all taking differing lengths of time, and all depending on the successful completion of earlier activities that it is almost impossible to work out when to do what. Most activities usually also have a very clearly defined time when they must be finished.

A CRITICAL PATH

If you work out the time taken by all the activities you will always find that there is one particular chain of events that determines the total time taken by the whole project. This is known as the *critical path*, and any delay or speeding up of this sequence of events will alter the time for the whole project. Other activities may possibly be delayed but this won't affect the overall time.

Working this out is no easy matter if you are limited to plotting the critical path by hand. But using your computer and the program given here, analysing the best possible situation becomes much simpler. The program lets you build up a database of all the activities required for a particular project along with the times they take (or estimates if they're not known) and the order they should be carried out. It then calculates the critical path and tells you how much slack time you have for the non-critical tasks—so you know which ones you have some leeway with and how long you can put off doing them without extending the whole project.

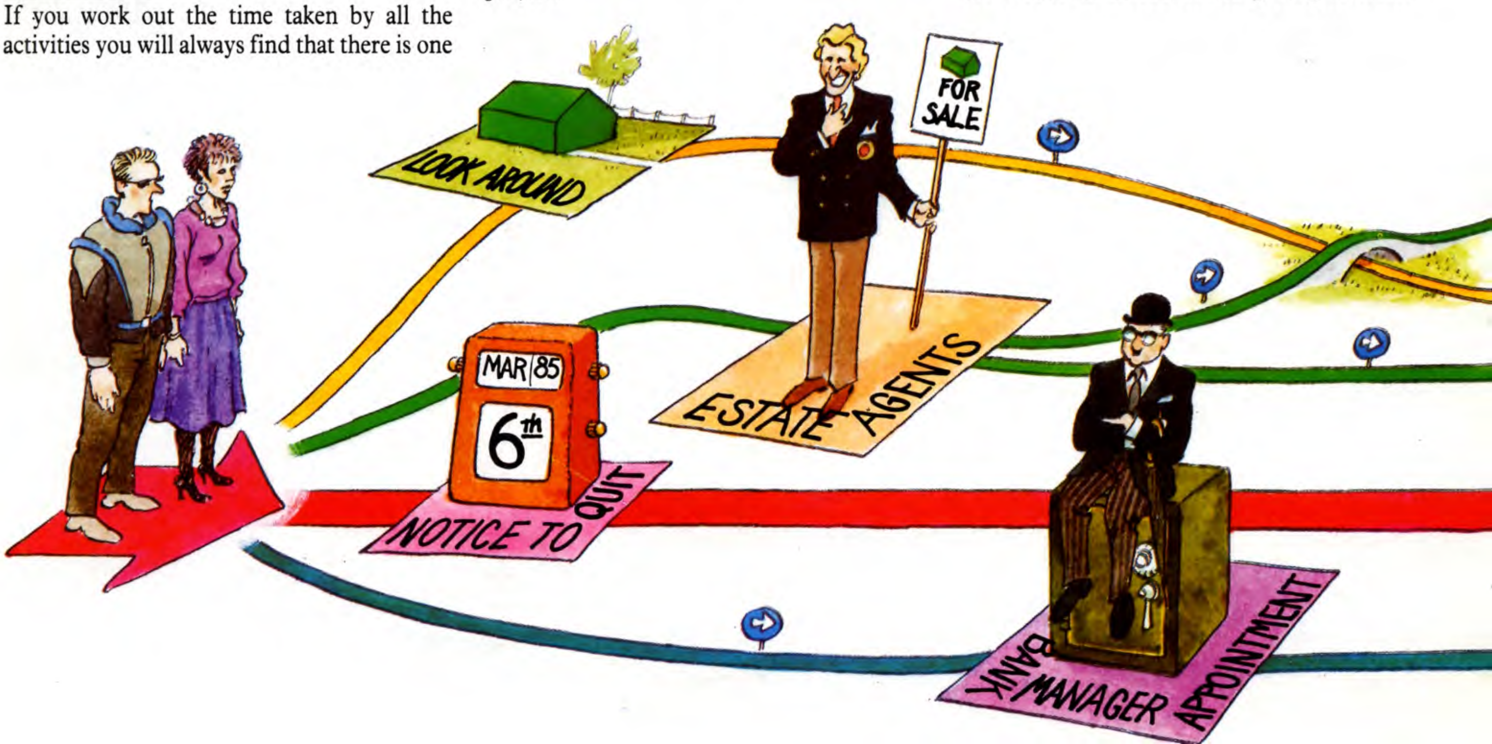
The program is given in two parts. The first part that accompanies this article sets up a database of all your activities while the second half, given next time, calculates the critical path so you can work out the quickest and most efficient way of carrying out your project.

TIME SAVING

Because it works out a critical path, the programming technique is often known as Critical Path Method (CPM) or Critical Path Analysis (CPA). Its other name is Program Evaluation and Review Technique or PERT, which refers to the method of planning using networks. These techniques first became fashionable in the early 1960s when they were used for the NASA space programme. One well quoted example were the savings made during the Polaris missile programme where the time taken for a given output was decreased by a factor of five! PERT programs are now used extensively in business and industry to increase efficiency in almost all projects. So if you own a small business (or even a large business) this program is ideal.

PLANNING A PROJECT

But its use is not limited to businesses—any project, however small, can be evaluated with the program. The program really becomes useful, though, when you are doing something more complicated than decorating a room or repairing a car. One project many people undertake is buying or selling a house, and this involves coordinating many different

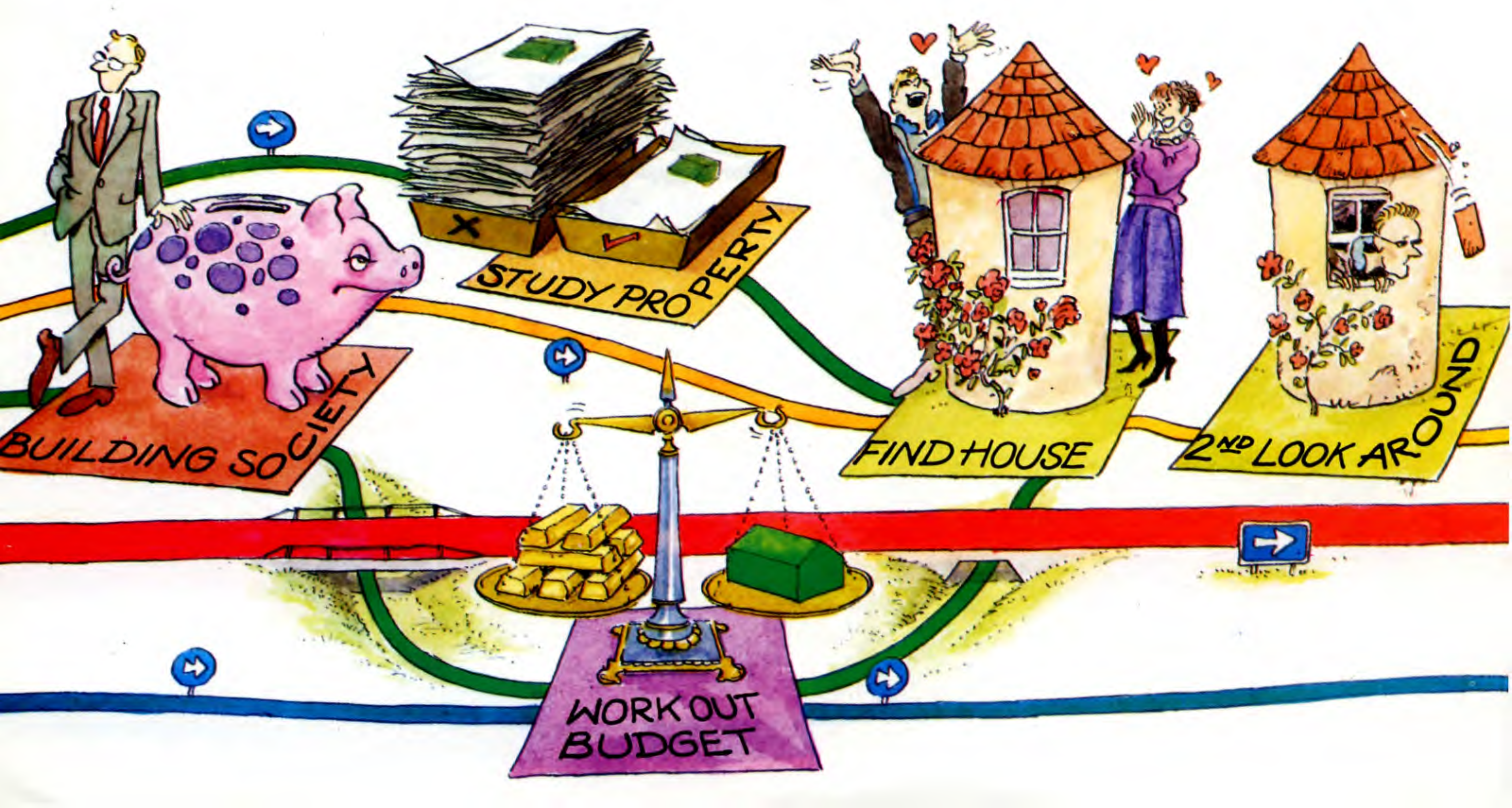


things—often within a very strict deadline. The diagram shows how you might draw a PERT chart for a house purchase. The circles enclose what are known as *events*. These are instances between activities that take up no time of their own. They simply mark the beginning or end of an activity. The activities are written along the lines joining the events, along with an estimate of the time needed.

At this stage many activities are uncertain and you'll find that a lot of the lines cross over one another. So although the information is all there, it is rather difficult to see exactly which are the important things to do quickly—or whether the whole thing can be done at all in the time allowed. Also as you progress with the purchase, other factors may appear which may alter the whole outlook (a rich aunt may leave you some money) so the chart would need to be updated. The advantages of using this program are enormous. You'll be able to create a neat printout of all the activities—*and* it can be updated in an instant. You'll be able to find the critical path through the mass of information so you can monitor this path more closely, taking immediate action if any delays appear. Or you may discover you need to replan some of the activities to make the project feasible in the first place.

Next time you'll see how to enter the information in the program. Meanwhile type in the first part of the program and save it ready to add the second part next time.

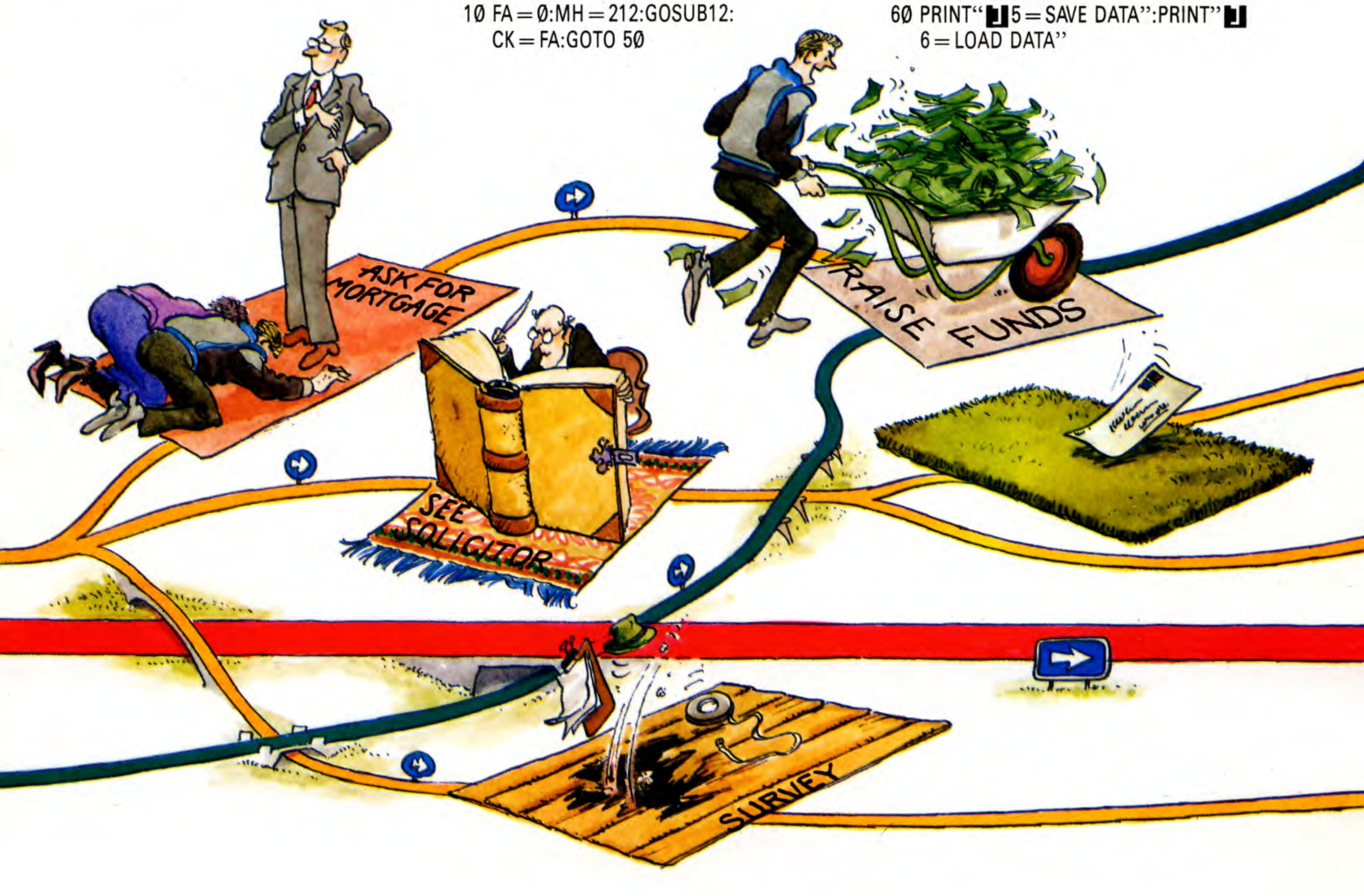
```
5 BORDER Ø: PAPER 4: INK Ø: CLS
7 POKE 23658,8: POKE 23609,2Ø
1Ø CLS : LET false=Ø: LET ma=1ØØ: LET
   me=1ØØ: LET mh=212: LET se=-1:
   LET fe=-1 : GOSUB 12: LET ck=false:
   LET aa=Ø: LET ee=Ø: GOTO 5Ø
12 LET zz=9999: LET true=1: LET
   p$="please input□": LET
   a$="□activity"
14 DIM w$(85,32): LET w$(1)="NO"+
   A$+"=PRECEDES event": LET w$(2)
   ="YOU CAN'T USE THIS MANY"
16 LET w$(3)="YOU CAN'T USE THIS
   NUMBER□": LET w$(4)=p$+"text for
   this□"
18 LET w$(5)=a$+" REFERS UNDEFINED
   EVENT "
22 DEF FN a(x)=x*(x<Ø): DEF FN
   z(x)=x*(x>Ø)
26 DIM a(ma): DIM g(ma)
3Ø DIM w(ma): DEF FN w(x)=ABS
   x*(x<1)+ABS (2-x)*(x>1):
    DEF FN x(x)=x*(2.37572+x*x*(15.94Ø2
   -x*x*(184.744-x*x*688.472)))/1.2Ø667
34 DEF FN l$(x)=(STR$ (x)+"□□□□
   □")(TO 6)
36 DEF FN b(x)=x-INT (x/256)*256
38 DEF FN p$(x)="--"( TO INT
   ((6-x)/2)): DEF FN q$(x)="□□"( TO
   INT ((6-x)/2))
4Ø DIM e(me)
42 DIM x(8)
44 DIM s(mh): DIM f(mh): DIM u(mh): DIM
   t(mh): DIM n(mh): DIM u$(mh,2Ø): DIM y
   (mh): DIM z(mh)
46 DIM p(mh): DIM q(mh)
48 DEF FN u(x)=u(ABS x+(x=Ø))*(x>Ø):
   RETURN
5Ø CLS : PRINT "1=define□";a$'"2=
   delete□";a$: PRINT "3=define event"'
   "4=delete event"
6Ø PRINT "5=save to tape"'"6=load
   tape"'"7=check tape"'"8=show details"
62 PRINT "9=QUIT"'"1Ø=check and sort
   network"
64 PRINT "11=calc with average durations"
66 PRINT "12=calc with uncertainties": PRINT
7Ø INPUT t: PRINT t: IF t=9 THEN STOP
72 IF t<1 OR t>12 THEN PRINT
   "t=";t;"□NOT UNDERSTOOD": GOTO 114
74 IF t>1Ø AND NOT ck THEN PRINT
   "PLEASE RUN DATA CHECK FIRST":
   GOTO 114
76 IF aa=Ø AND (t>7 OR t=5) THEN
   PRINT "CAN'T DO - NO";a$;
   "□INPUT": GOTO 114
8Ø IF t>7 THEN GOTO 1ØØ
82 IF t=6 THEN CLEAR : LET t=6
84 IF t>4 THEN PRINT "please input file
   name:";: INPUT f$: PRINT f$: GOTO 1ØØ
86 LET f$=a$: IF t>2 THEN LET
   f$="event"
88 PRINT p$;f$;"□number": PRINT "or
   zero to quit□";
9Ø INPUT u: PRINT u: LET u=INT u: IF u=Ø
   THEN GOTO 5Ø
```



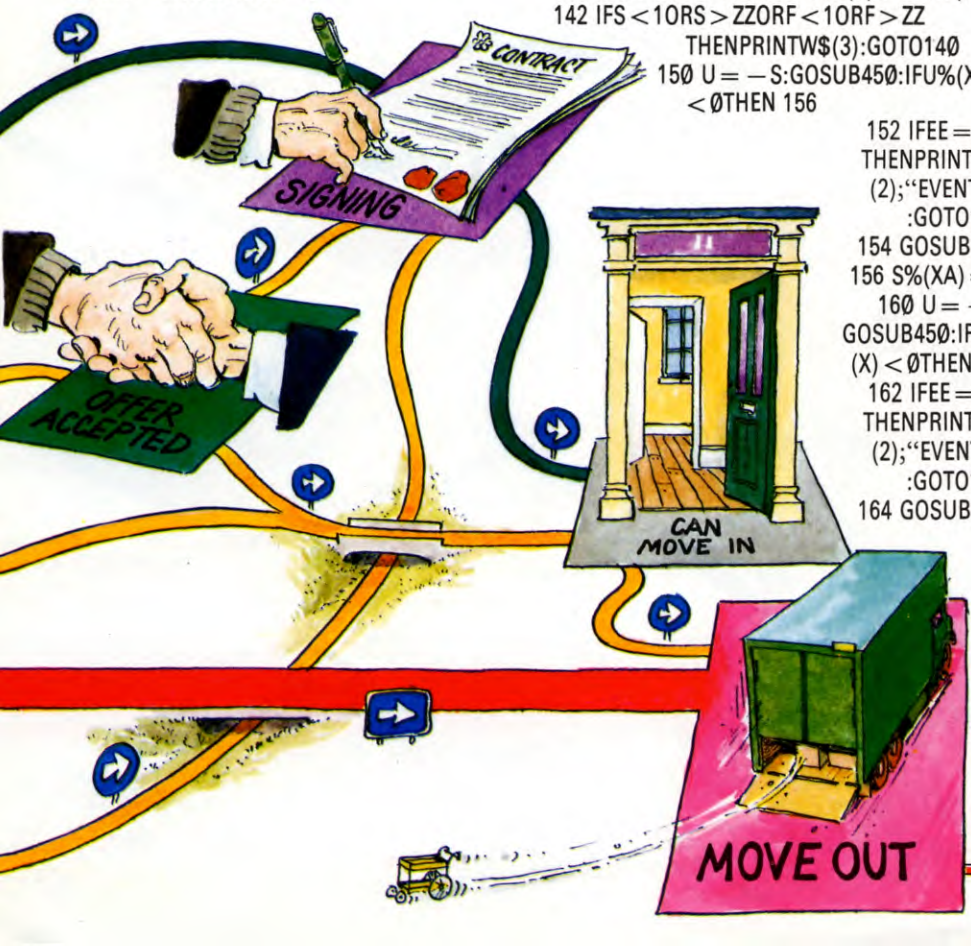
```
92 IF u<1 OR u>zz THEN PRINT w$(3):
   GOTO 88
94 IF t>2 THEN LET u=-u
96 GOSUB 45Ø: LET ck=false
98 IF (t=2 OR t=4) AND (Ø=u(x) OR
   zz<u(x)) THEN PRINT "YOU NEVER USED
   THIS NUMBER": GOTO 114
1ØØ GOSUB 2Ø*(t=1)+1ØØ*t+9ØØ*
   (t-1Ø)*(t>1Ø)
112 GOTO 5Ø
114 FOR t=1 TO 5ØØ: NEXT t: GOTO 5Ø
12Ø IF Ø<u(x) AND zz>=u(x) THEN
   GOSUB 942: GOSUB 932: GOTO 13Ø
122 IF aa=ma THEN PRINT w$(2);f$:
   RETURN
124 LET aa=aa+1: LET a(aa)=x: LET
   u(x)=u
13Ø PRINT w$(4);f$;":";: INPUT u$(x):
   PRINT u$(x): LET xa=x
14Ø PRINT p$;"start event, finish event□";:
   INPUT s,f: PRINT s;"□";f: LET s=INT s:
   LET f=INT f
142 IF s<1 OR s>zz OR f<1 OR f>zz
   THEN PRINT w$(3): GOTO 14Ø
15Ø LET u=-s: GOSUB 45Ø: IF u(x)<Ø
   THEN GOTO 156
152 IF ee=me THEN PRINT w$(2);"events":
   GOTO 14Ø
154 GOSUB 35Ø
156 LET s(xa)=x
16Ø LET u=-f: GOSUB 45Ø: IF u(x)<Ø
   THEN GOTO 166
162 IF ee=me THEN PRINT w$(2);"events":
   GOTO 14Ø
164 GOSUB 35Ø
166 LET f(xa)=x
17Ø PRINT p$;"probable time to do□";:
   INPUT t(xa): PRINT t(xa)
172 IF t(xa)<Ø THEN PRINT "YOU CAN'T DO
   IT THIS FAST": GOTO 17Ø
18Ø PRINT "input time you are 9Ø% certain":
   PRINT "it can be done in□";: INPUT
   n(xa): PRINT n(xa)
182 IF n(xa)<t(xa) THEN PRINT "THIS
   DISAGREES WITH PROBABLE TIME":
   GOTO 17Ø
19Ø RETURN
2ØØ FOR b=1 TO aa: IF x=a(b) THEN LET
   a=b
22Ø NEXT b: LET a(a)=a(aa): LET
   u(x)=zz+1: LET aa=aa-1: RETURN
3ØØ IF u(x)<Ø THEN GOSUB 946: GOSUB
   933: GOTO 33Ø
31Ø IF ee=me THEN PRINT w$(2);f$:
   RETURN
312 GOSUB 35Ø
```

C=

```
5 POKE 5328Ø,12:POKE 53281,12
1Ø FA=Ø:MH=212:GOSUB12:
   CK=FA:GOTO 5Ø
12 CLS$="♥":CM$=",":ZZ=32766:
   MA=1ØØ:ME=1ØØ:QT$=CHR$(34):TR
   =-1:PRINTCLS$
14 P$="PLEASE INPUT□":A$="ACTIVITY"
16 DIMW$(5):W$(1)="NO□"+A$+"□
   PRECEDES EVENT":W$(2)="YOU CANT
   USE THIS MANY "
18 W$(3)="YOU CAN'T USE THIS NUMBER"
   :W$(4)=P$+"TEXT FOR THIS □"
2Ø W$(5)=A$+"□REFERS TO UNDEFINED
   EVENT"
22 DEFFNA(X)=-X*(X<Ø):DEFFNZ(X)
   =-X*(X>Ø)
28 DIMA%(MA),G%(MA)
3Ø DIMW(MA):DEFFNW(X)=-ABS(X)*(X
   <=1)-ABS(2-X)*(X>1)
32 DEFFNX(X)=X*(2.37572+X*X*(15.94Ø2
   -X*X*(184.744-X*X*688.472)))/1.2Ø667
4Ø DIM E%(ME)
44 DIM S%(MH),F%(MH),U%(MH),T(MH),
   N(MH),U$(MH),Y(MH),Z(MH)
46 DIM P(MH),Q(MH)
48 DEFFNU(XX)=TR*U%(ABS(XX)
   -(XX=Ø))*(XX>Ø):RETURN
5Ø PRINTCLS$TAB(1Ø)"■■■MAIN
   MENU■■●"
53 PRINT"■1=DEFINE■";A$:PRINT"■2
   =DELETE■";A$
55 PRINT"■3=DEFINE EVENT":PRINT"■4
   =DELETE EVENT"
6Ø PRINT"■5=SAVE DATA":PRINT"■
   6=LOAD DATA"
```



```
61 PRINT"■7=DELETE FILE FROM DISK":PRINT"■8=SHOW DETAILS"
62 PRINT"■9=(RESTART)"
63 PRINT"1Ø=CHECK AND SORT NETWORK"
64 PRINT"11=CALC WITH AVERAGE DURATIONS"
65 PRINT"12=CALC WITH UNCERTAINTIES ":PRINT"13=OUTPUT TO □";
66 IF KK$="Y" THEN PRINT"■ PRINTER■/SCREEN"
67 IF KK$<>"Y" THEN PRINT"PRINTER/■SCREEN"
68 PRINT"14=(QUIT)"
69 T=Ø:INPUT"■■■■ENTER OPTION";T
7Ø IFT=14THENINPUT"ARE YOU SURE (Y/N)";AN$:IFAN$="Y"THENSYS58648:END
71 IF T=13 OR T=14 THEN 1ØØ
72 IF T=9 THEN 9ØØ
73 IFT<1ORT>13THEN PRINT"■■■■CODE"□;T;" NOT UNDERSTOOD":GOTO114
74 IFT>1ØANDNOT(CK)THEN PRINT"■■■■PLEASE RUN DATA CHECK FIRST":GOTO114
76 IFAA=ØAND(T>7ORT=5)THENPRINT"■■■■CAN'T DO — NO□"+A$+" □INPUT":GOTO114
8Ø IFT>7THEN1ØØ
82 IFT=6THENCLR:T=6
84 IFT>4THENPRINT"■"P$;"■FILE NAME":INPUTF$:GOTO1ØØ
86 F$=A$:IFT>2THENF$="EVENT"
88 PRINTCL$"PLEASE INPUT■";F$;"■NUMBER":PRINT"OR ZERO TO QUIT"
9Ø INPUTU:U=INT(U):IFU=ØTHEN5Ø
92 IFU<1OR(U>ZZ)THENPRINTW$(3):FORDE=1TO999:NEXTDE:GOTO88
94 IFT>2THENU=-U
96 GOSUB45Ø:CK=FA
98 IF(T=2ORT=4)AND(Ø=U%(X)ORZZ<U%(X))THENPRINT"NUMBER NOT USED":GOTO114
1ØØ IF KK$="Y" AND (T>7 AND T<12)THEN OPEN 4,4:CMD4
1Ø1 PRINTCL$:ONTGOSUB12Ø,2ØØ,3ØØ,4ØØ,5ØØ,6ØØ,7ØØ,8ØØ,9ØØ,1ØØØ,2ØØØ,3ØØØ,96Ø
1Ø5 IF KK$="Y" AND(T>7 AND T<13) THEN PRINT#4:CLOSE4
11Ø IFT<5THEN86
112 GOTO5Ø
114 FORT=1TO1ØØØ:NEXTT:GOTO5Ø
12Ø IFØ<U%(X)ANDZZ>=U%(X)THEN GOSUB942:GOSUB932:GOTO13Ø
122 IF(AA=MA)THENPRINTW$(2);F$:RETURN
124 AA=AA+1:A%(AA)=X:U%(X)=U
13Ø PRINTW$(4);F$:INPUTU$(X):XA=X
14Ø PRINTP$;"START EVENT, FINISH EVENT":INPUT S,F:S=INT(S):F=INT(F)
142 IFS<1ORS>ZZORF<1ORF>ZZ THENPRINTW$(3):GOTO14Ø
15Ø U=-S:GOSUB45Ø:IFU%(X)<ØTHEN 156
152 IFEE=ME THENPRINTW$(2);"EVENTS":GOTO14Ø
154 GOSUB35Ø
156 S%(XA)=X
16Ø U=-F:GOSUB45Ø:IFU%(X)<ØTHEN166
162 IFEE=ME THENPRINTW$(2);"EVENTS":GOTO14Ø
164 GOSUB35Ø
166 F%(XA)=X
17Ø PRINTP$;"PROBABLE TIME TO DO":INPUTT(XA)
172 IFT(XA)<ØTHENPRINT"YOU CANT DO IT THIS FAST":GOTO17Ø
18Ø PRINTP$;"TIME YOU ARE 9Ø% CERTAIN":PRINT"IT CAN BE DONE IN":INPUTN(XA)
182 IFN(XA)<T(XA)THENPRINT"THIS DISAGREES WITH PROBABLE TIME":GOTO17Ø
19Ø RETURN
2ØØ FORB=1TOAA:IFA%(B)=XTHENA=B
22Ø NEXTB:XX=A%(AA):U%(X)=ZZ+1:A%(A)=XX:AA=AA-1:RETURN
3ØØ IFU%(X)<ØTHENPRINT"■EVENTS":XP=U%(X):GOSUB95Ø:PRINTU$(X):GOTO33Ø
31Ø IF(EE=ME)THENPRINTW$(2);F$: RETURN
312 GOSUB35Ø
33Ø PRINTW$(4);F$:INPUTU$(X):S%(X)=Ø:RETURN
35Ø EE=EE+1:E%(EE)=X:S%(X)=-1:F%(X)=Ø:U%(X)=U
36Ø T(X)=Ø:N(X)=Ø:U$(X)="":RETURN
4ØØ FORF=1TOEE:IFE%(F)=XTHENE=F
42Ø NEXTF:XX=E%(EE)::U%(X)=ZZ+1:E%(E)=XX:EE=EE-1:RETURN
45Ø Z=U-INT((U-1)/MH)*MH:Y=2:X=Ø
46Ø IFX=ØTHENIFØ=U%(Z)ORZZ+1=U%(Z)THENX=Z
47Ø IFU=U%(Z)THENX=Z:RETURN
48Ø IFY=1ORØ=U%(Z)THENRETURN
49Ø Z=Z+Y-MH*INT((Z+Y-1)/MH):Y=Y+Y-MH*INT((Y+Y-1)/MH):GOTO46Ø
5ØØ OPEN1,8,8,"Ø:"+F$+",S,W":PRINT#1,MA;CM$;ME;CM$;MH;CM$;AA;CM$;EE;CM$;CK
51Ø IFCKTHENPRINT#1,SE;CM$;FE
52Ø FORA=1TOAA:X=A%(A):PRINT#1,X;CM$U%(X)CM$S%(X)CM$F%(X)CM$T(X)CM$N(X)CM$;
53Ø PRINT#1,G%(A)CM$QT$U$(X)QT$:NEXTA
54Ø FORE=1TOEE:X=E%(E):PRINT#1,X;CM$U%(X)CM$S%(X)CM$F%(X)CM$T(X)CM$N(X)
55Ø PRINT#1,QT$U$(X)QT$:NEXTE
56Ø FORX=1TOMH:IFU%(X)=ZZ+1THEN PRINT#1,X
57Ø NEXTX:PRINT#1,Ø
58Ø CLOSE1:RETURN
```




```
1 MODE6
1Ø MH=1Ø1:MA=5Ø:ME=5Ø:GOSUB2Ø:CK=FALSE:AA=Ø:EE=Ø:GOTO11Ø
2Ø UU=Ø:ZZ=9999
3Ø A$="□ACTIVITY"
4Ø DIMW$(5):W$(1)="NO□"+A$
```

```
   +"□PRECEDES EVENT□":W$(2)
   ="YOU CANT USE THIS MANY□"
50 W$(3)="YOU CANT USE THIS
   NUMBER":W$(4)="INPUT TEXT FOR
   THIS□"
60 W$(5)=A$+"□REFERS TO UNDEFINED
   EVENT□"
70 DIMA□MA,G□MA,E□ME,S□MH,F□MH
80 DIMW(MA)
90 DIMU%(MH),T(MH),N(MH),U$(MH)
   ,Y(MH),Z(MH),P(MH),Q(MH)
100 RETURN
110 CLS:PRINT"'□1=DEFINE,
   2=DELETE";A$'"□3=DEFINE,
   4=DELETE EVENT"
120 PRINT"□5=SAVE INFORMATION,
   6=LOAD INFORMATION"'"□7=SHOW
   DETAILS"
130 PRINT"□8=CALC WITH AVERAGE
   DURATIONS"'"□9=CALC WITH
   UNCERTAINTIES"'"□13=QUIT"
140 INPUTT:IFT=13THENEND
150 IFT>10ORT<1THENPRINT"CODE□";
   T;"□NOT UNDERSTOOD":GOTO280
160 IFT>6THEN260
170 IFT=6THENCLEAR:T=6
180 IFT=5ORT=6THENINPUT"PLEASE
   INPUT FILE NAME",F$:GOTO260
190 F$=A$:IFT>2THENF$="EVENT"
200 PRINTF$;"□NUMBER"'"OR ZERO TO
   QUIT"
210 INPUTU:U=INT(U):IFU=ØTHEN110
220 IFU<1OR(U>ZZ)THENPRINT'W$(3):
   GOTO200
230 IFT>2THENU=-U
240 GOSUB580:CK=FALSE
250 IF(T=2OR T=4)AND(Ø=U%(X)OR
   ZZ<U%(X))THENPRINT "YOU HAVEN'T
   USED THIS NUMBER":GOTO280
260 ON(T)GOSUB290,480,500,560,630,700,
   760,1360,1480
270 IFT<5THEN190
280 GOSUB840:GOTO110
290 JM=3:IFU%(X)>ØANDZZ>=U%(X)
   THENCLS:GOSUB850:GOSUB830:GOTO
   320
300 IF(AA=MA)THENPRINTW$(2);F$:
   RETURN
310 AA=AA+1:A?AA=X:U%(X)=U
320 PRINTW$(4);F$:INPUTU$(X):XA=X
330 PRINT"START EVENT,FINISH EVENT":
   INPUTS%,F%
340 IFS%=F%ORS%<1OR(S%>ZZ)OR
   F%<1OR(F%>ZZ)THENPRINTW$(3):
   GOTO330
350 U=-S%:GOSUB580:IFU%(X)<ØTHEN
   380
360 IF(EE=ME)THENPRINTW$(2);"EVENTS":
   GOTO330
370 GOSUB540
380 S?XA=X
390 U=-F%:GOSUB580:IFU%(X)<ØTHEN
   420
400 IF(EE=ME)THENPRINTW$(2);"EVENTS"
   :GOTO330
410 GOSUB540
420 F?XA=X
430 PRINT"PROBABLE TIME TO DO":INPUTT
   (XA)
440 IFT(XA)<ØTHENPRINT"YOU CAN'T DO
   IT THIS FAST":GOTO430
450 INPUT"TIME YOU ARE 90%
   CERTAIN""IT CAN BE DONE IN",N(XA)
460 IFN(XA)<T(XA)THENPRINT"THIS
   DISAGREES WITH PROBABLE TIME":
   GOTO430
470 RETURN
480 FORB=1TOAA:IF(X=A?B)THENA%=B
490 NEXTB:A?A%=A?AA:U%(X)=ZZ+1:
   AA=AA-1:RETURN
```

```
10 PCLEAR1:CLEAR2000:MH=212:ME=
   100:MA=100:FA=Ø:GOSUB20:CK=FA:
   GOTO140
20 ZZ=9999:TR=-1:P$="INPUT□":
   A$="□ACTIVITY":E$=CHR$(13)
30 DIMW$(5):W$(1)="NO"+A$+
   "□PRECEDES EVENT":W$(2)="YOU
   CAN'T USE THIS MANY□"
40 W$(3)="YOU CAN'T USE THIS
   NUMBER":W$(4)=P$+"TEXT FOR
   THIS□"
50 W$(5)="REFERS TO UNDEFINED EVENT"
60 DEFFNA(X)=-X*(X<Ø):DEFFNZ(X)=
   -X*(X>Ø)
70 DIMA(MA),G(MA)
80 DIMW(MA):DEFFNW(X)=-ABS(X)*
   (X<=1)-ABS(2-X)*(X>1)
90 DEFFNX(X)=X*(2.37572+X*X*(15.9402
   -X*X*(184.744-X*X*688.472)))/1.20667
100 DIME(ME)
110 DIMS(MH),F(MH),U(MH),T(MH),N(MH),
   U$(MH),Y(MH),Z(MH)
120 DIMP(MH),Q(MH)
130 DEFFNU(X)=-U(ABS(X)-(X=Ø))*
   (X>Ø):RETURN
140 CLS:PR=Ø:PRINT@11,"MAIN MENU":
   PRINT" 1=DEFINE";A$;"□OR EVENT":
   PRINT" 2=DELETE";A$;"□OR EVENT"
150 PRINT" 3=SAVE DATA":PRINT
   " 4=LOAD DATA":PRINT" 5=PRINT
   DETAILS":PRINT" 6=QUIT"
160 PRINT" 7=CHECK AND SORT
   NETWORK"
170 PRINT" 8=CALC WITH AVERAGE
   DURATIONS"
180 PRINT" 9=CALC WITH
   UNCERTAINTIES":PRINT" ?";
190 T$=INKEY$:IFT$<"1"ORT$>"9"
   THEN190
200 T=VAL(T$):PRINTT
210 IFT>7ANDNOT(CK)THENPRINT" RUN
   DATA CHECK (7) FIRST":GOTO380
220 IFAA=ØAND(T>4ORT=3)THENPRINT
   "CAN'T DO - NO";A$;"□INPUT":GOTO
   380
230 IFT>4THEN350
240 IFT=4THENCLEAR2000:T=4
250 IFT>2THENPRINT"PLEASE INPUT FILE
   NAME":INPUTF$:GOTO350
260 CLS:PRINTA$;"□OR EVENT (A/E) ? ";
270 T$=INKEY$:IFT$<>"A"ANDT$<>
   "E"THEN270
280 PRINTT$:IFT$="A"THENF$=A$ELSE
   F$="□EVENT"
290 PRINT:PRINTP$;F$;"□NUMBER":PRINT
   "OR ZERO TO QUIT"
300 INPUTU:U=INT(U):IFU=ØTHEN140
310 IFU<1ORU>ZZ THENPRINTW$(3):
   GOTO280
320 IFT$="E"THENU=-U
330 GOSUB700:CK=FA
340 IF(T=2ORT=4)AND(Ø=U(X)ORZZ<U
   (X))THENPRINT"YOU NEVER USED THIS
   NUMBER":GOTO380
350 ONT GOSUB390,590,750,830,890,960,
   1070,1550,1660
360 IFT<3THEN290
370 GOTO140
380 FORT=1TO1000:NEXTT:GOTO140
390 IFF$<>A$THEN620
400 IFØ<U(X)ANDZZ>=U(X)GOSUB1030:
   GOSUB1000:GOTO430
410 IFAA=MA THENPRINTW$(2);F$:
   RETURN
420 AA=AA+1:A(AA)=X:U(X)=U
430 PRINTW$(4);F$:INPUTU$(X):XA=X
440 PRINTP$;"START EVENT, FINISH
   EVENT":INPUTS,F:S=INT(S):F=INT(F)
450 IFS<1ORS>ZZ ORF<1ORF>ZZ THEN
   PRINTW$(3):GOTO440
460 U=-S:GOSUB700:IFU(X)<ØTHEN490
470 IFEE=ME THENPRINTW$(2);"EVENTS":
   GOTO440
480 GOSUB660
490 S(XA)=X
500 U=-F:GOSUB700:IFU(X)<ØTHEN530
510 IFEE=ME THENPRINTW$(2);"EVENTS":
   GOTO440
520 GOSUB660
530 F(XA)=X
540 PRINTP$;"PROBABLE TIME TO DO":
   INPUTT(XA)
550 IFT(XA)<ØTHENPRINT"YOU CAN'T DO
   IT THIS FAST":GOTO540
560 PRINT"INPUT TIME YOU ARE 90%
   CERTAIN":PRINT"IT CAN BE DONE IN":
   INPUTN(XA)
570 IFN(XA)<T(XA)THENPRINT"THIS
   DISAGREES WITH PROBABLE TIME":
   GOTO540
580 RETURN
```

FORMS OF THE NATURAL WORLD

Move on from the idealized shapes of mathematically generated fractals to forms which are capable of modelling the natural world with convincing reality

In the first article on fractals, you saw how simple recursive programs can be used to generate fascinating shapes by the repeated application of a single operation. These mathematically generated patterns display order and symmetry, but although they also share many features with the irregular forms found in nature, they still seem like little more than interesting curiosities.

Although fractals like these come much closer to modelling the natural world than is possible using the perfect shapes of traditional science, so far, they still fall a long way short of realism. This article shows how fractals can model forms found in nature.

The first program draws one of the natural world's most symmetrical forms—the six sided shape of a snowflake.

```
10 BORDER Ø:PAPER Ø:INK 5:BRIGHT 1:CLS
20 LET AN=2*ATN (1)/3:LET S2=2/SQR (3)
30 LET XC=127:LET YC=9Ø:LET
   S=12Ø:LET C=2
50 GOSUB 1ØØØ
60 STOP
1ØØØ LET S=S/3:IF S<1 THEN LET
   S=S*3:RETURN
1Ø2Ø PLOT INVERSE 1;OVER 1;INT
   (XC+S2*S*SIN (-AN)),(YC-S2*S*COS
   (-AN)):FOR K=Ø TO 8*ATN (1)-AN
   STEP 2*AN
1Ø3Ø DRAW XC+2*S*SIN (K)-PEEK
   23677,YC-2*S*COS (K)-PEEK 23678
1Ø4Ø DRAW XC+S2*S*SIN (K+AN)-PEEK
   23677,YC-S2*S*COS (K+AN)-PEEK
   23678
1Ø5Ø NEXT K
1Ø6Ø LET C=C-1:GOSUB 1ØØØ
1Ø7Ø LET YC=YC-1.36*S:GOSUB 1ØØØ
1Ø8Ø LET YC=YC+.68*S:LET
   XC=XC+1.19*S: GOSUB 1ØØØ
1Ø9Ø LET YC=YC+1.36*S:GOSUB 1ØØØ
11ØØ LET YC=YC+.68*S:LET
   XC=XC-1.19*S:GOSUB 1ØØØ
111Ø LET YC=YC-.68*S:LET
   XC=XC-1.19*S:GOSUB 1ØØØ
112Ø LET YC=YC-1.36*S:GOSUB 1ØØØ
113Ø LET YC=YC+.68*S: LET
   XC=XC+1.19*S:LET S=S*3:LET
   C=C+1:RETURN
```

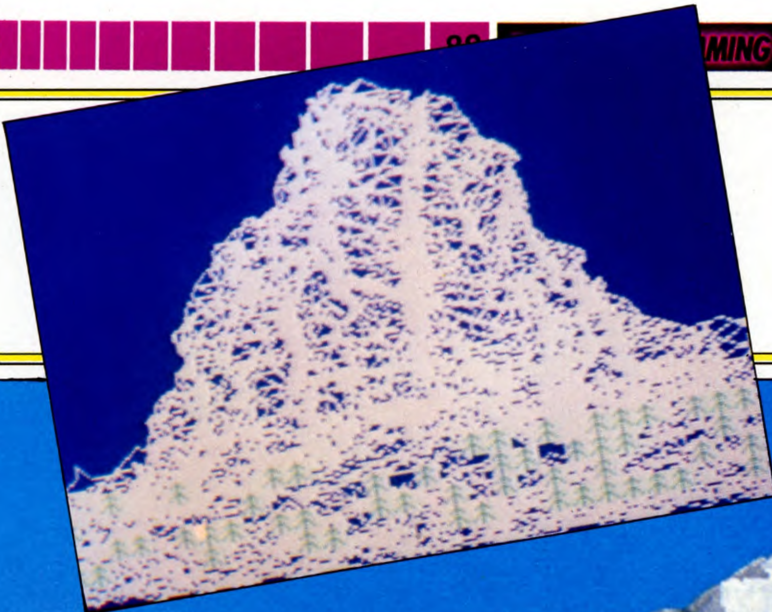
```
1Ø HIRES Ø,1:MULTI Ø,1,7:COLOUR 5,5
2Ø AN=2*ATN(1)/3:S2=2/SQR(3)
3Ø XC=8Ø:YC=99:S=99:C=4
5Ø GOSUB 1ØØØ:FOR D=1 TO 5ØØØ:NEXT
   D:COLOUR Ø,Ø
6Ø MULTI RND(1)*16,RND(1)*16,RND(1)*16:
   GOTO 6Ø
1ØØØ S=S/3:IF S<1 THEN S=S*3:RETURN
1Ø1Ø CL=CL+1:IF CL>3 THEN CL=1
1Ø2Ø XX=INT(XC+S2*S*SIN(-AN)):YY=
   INT(YC-S2*S*COS(-AN))
1Ø25 FOR K=Ø TO 8*ATN(1)-AN STEP 2*AN
1Ø3Ø LINE XX,YY,XC+2*S*SIN(K),YC-2*S*
   COS(K),CL
1Ø4Ø LINE XC+2*S*SIN(K),YC-2*S*COS(K),
   XC+S2*S*SIN(K+AN),YC-S2*S*COS
   (K+AN),CL
1Ø45 XX=XC+S2*S*SIN(K+AN):YY=YC-
   S2*S*COS(K+AN)
1Ø5Ø NEXT K
1Ø6Ø C=C-1:GOSUB 1ØØØ
1Ø7Ø YC=YC-1.36*S:GOSUB 1ØØØ
1Ø8Ø YC=YC+.68*S:XC=XC+1.19*S:
   GOSUB 1ØØØ
1Ø9Ø YC=YC+1.36*S:GOSUB 1ØØØ
11ØØ YC=YC+.68*S:XC=XC-1.18*S:
   GOSUB 1ØØØ
111Ø YC=YC-.68*S:XC=XC-1.19*S:
   GOSUB 1ØØØ
112Ø YC=YC-1.36*S:GOSUB 1ØØØ
113Ø YC=YC+.68*S:XC=XC+1.19*S:S=
   S*3:C=C+1:RETURN
```

The listing is as for the Commodore 64, except for the following lines:

```
1Ø GRAPHIC 2:COLOR Ø,Ø,1,1
2Ø AN=ATN(1)/3:SX=2/SQR(3)
3Ø XC=512:YC=512:S=7ØØ
5Ø GOSUB 1ØØØ
6Ø GOTO 6Ø
1ØØØ S=S/3:IF S<1Ø THEN S=S*3:
   RETURN
1Ø1Ø CL=CL+1:IF CL>7 THEN CL=1
1Ø2Ø POINT Ø,INT(XC+S2*S*SIN(-AN)),
   INT(YC-S2*S*COS(-AN)):REGION (CL)
1Ø3Ø DRAW 1 TO XC+2*S*SIN(K),YC-2*S*
   COS(K)
1Ø4Ø DRAW 1 TO XC+S2*S*SIN(K+AN),
   YC-S2*S*COS(K+AN)
```

```
1Ø MODE1
2Ø VDU23;82Ø2;Ø;Ø;Ø;19,1,4;Ø;19,2,6;Ø;
3Ø PROCSTAR(64Ø,512,5ØØ,Ø)
4Ø END
5Ø DEFPROCSTAR(X,Y,S,C)
6Ø LOCAL I
7Ø IF S<16 THEN ENDPROC
8Ø IF C=4 THEN C=1
9Ø GCOLØ,C
1ØØ VDU29,X;Y;
11Ø XL=S*COS(PI/6):YL=S/2
12Ø MOVEØ,S:MOVEXL,-YL:PLOT85,-XL,
   -YL
13Ø MOVEØ,-S:MOVEXL,YL:PLOT85,-XL,
   YL
14Ø PROCSTAR(X,Y,S/3,C+1)
15Ø FORI=ØTO2*PI-PI/3STEPPI/3
16Ø PROCSTAR(X+S*SIN(I)*.68,Y+S*COS
   (I)*.68,S/3,C+2)
17Ø NEXT
18Ø ENDPROC
```

```
1Ø PMODE3,1:PCLS:SCREEN1,Ø
2Ø AN=2*ATN(1)/3:S2=2/SQR(3)
3Ø XC=127:YC=95:S=135:C=4
5Ø GOSUB1ØØØ
6Ø GOTO6Ø
1ØØØ S=S/3:IF S<1 THEN S=S*3:
   RETURN
1Ø1Ø IF C=2 THEN COLOR4 ELSEIFC=1
   THENCOLOR2 ELSECOLORC
1Ø2Ø DRAW"BM"+STR$(INT(XC+S2*S*SIN
   (-AN)))+","+STR$(INT(YC-S2*S*
   COS(-AN))):FORK=ØTO8*ATN(1)-AN
   STEP 2*AN
1Ø3Ø LINE-(XC+2*S*SIN(K),YC-2*S*COS
   (K)),PSET
1Ø4Ø LINE-(XC+S2*S*SIN(K+AN),YC-
   S2*S*COS(K+AN)),PSET
1Ø5Ø NEXT:PAINT(XC,YC)
1Ø6Ø C=C-1:GOSUB1ØØØ
1Ø7Ø YC=YC-1.36*S:GOSUB1ØØØ
1Ø8Ø YC=YC+.68*S:XC=XC+1.19*S:
   GOSUB1ØØØ
1Ø9Ø YC=YC+1.36*S:GOSUB1ØØØ
11ØØ YC=YC+.68*S:XC=XC-1.19*S:
```



- MODELS OF SYMMETRY
- SNOWFLAKE PROGRAM
- MODELS OF UNCERTAINTY
- COMPUTER MOUNTAIN
- SHAPE GENERATOR PROGRAM

Advanced fractal techniques are responsible for some of the most convincing images yet generated using computer graphics. The large mountainscape shown here exists only in the memory of a computer, but could easily pass for the real thing.

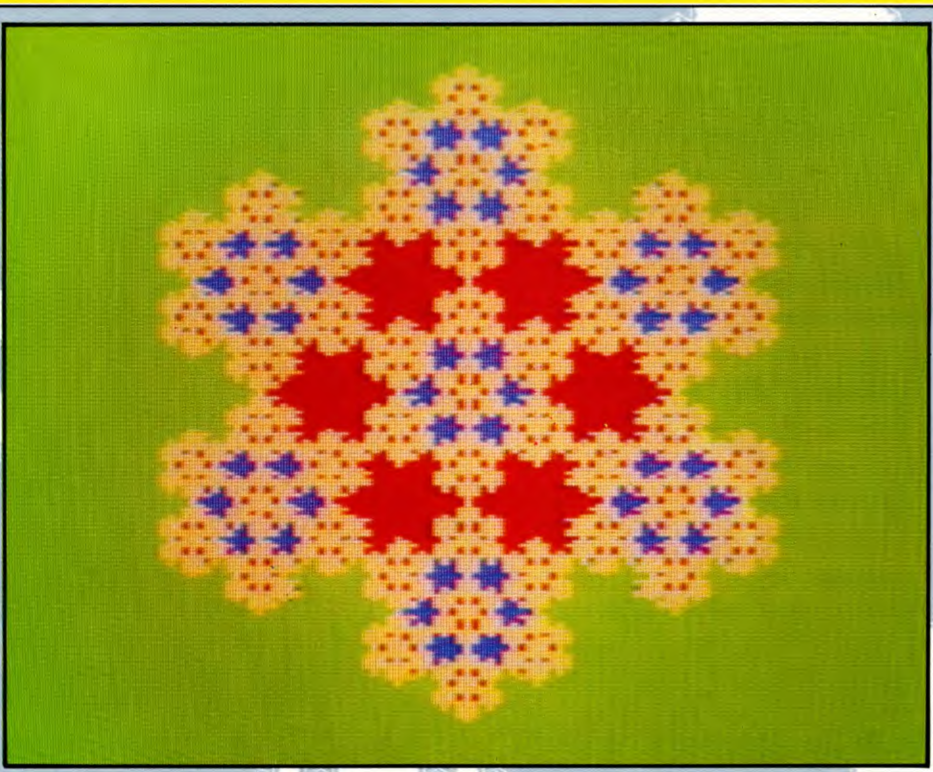
An image of this complexity takes masses of processing time on huge, specialized computers, but similar techniques can be applied on home micros as shown by the Spectrum mountain image.

```
    GOSUB1ØØØ
111Ø YC=YC-.68*S:XC=XC-1.19*S:
    GOSUB1ØØØ
112Ø YC=YC-1.36*S:GOSUB1ØØØ
113Ø YC=YC+.68*S:XC=XC+1.19*S:S=
    S*3:C=C+1:RETURN
```

This program is based on the snowflake curve originally drawn by von Koch. It can be regarded as either an infinitely crinkled snowflake or an island of infinite coastline. The program specifies (Line 2Ø) an equilateral triangle (in which all three angles are 60° and all sides are equal), with a size factor set to S2. Line 3Ø sets up X and Y coordinates for the centre, a scale factor for the size of each triangle and a variable to vary the colour. Line 5Ø calls a recursive subroutine, which draws a six-sided star-shaped figure.

SYMMETRY AND CHAOS

Despite the outline's irregularity, much of the symmetry of the star-shape remains. Symmetry is necessary for modelling shapes like the snowflake, which combine order and chaos, and such shapes are common in Nature. But many natural structures that can best be understood as fractals are totally lacking in symmetry. Examples of these are the bends of the Mississippi River, the surface of a soapflake, the holes in a Swiss cheese, the craters of the Moon, the veins and arteries of the body and the shapes of mountains. What distinguishes these from the symmetrical, mathematically generated shapes is that they also contain a degree of randomness. But it is possible to generate this, too, by using the computer's own random number generator. Here, for example, is a program to model a mountain:



```
1Ø BORDER Ø:PAPER Ø:INK 7:BRIGHT 1:CLS
15 PRINT AT 6,2;INVERSE 1;
   "□FRACTAL MOUNTAIN GENERATOR□"
2Ø DIM C(2ØØ,2,2):LET F=1:LET G=2:LET
   C(1,1,2)=25:LET C(1,1,1)=Ø
22 INPUT "ENTER 'RESOLUTION' OF
   MOUNTAIN [ 16-1ØØ ] ?□";S
23 IF S<16 OR S>1ØØ THEN GOTO 22
24 INPUT "ENTER DEGREE OF
   'RUGGEDNESS' YOU REQUIRE [1 TO
   5 ] ?□";RG
25 IF RG<1 OR RG>5 THEN GOTO 24
26 DEF FN R(X)=RG-((RND*X)*(2*RG))
27 PAPER 1:CLS
3Ø LET L=23Ø/S:LET H=L/(SQR 3)
4Ø FOR K=2 TO S+1:LET C(K,1,1)=C
   (1,1,1)+L*K-FN R(1):LET C(K,1,2)=
   C(K-1,1,2)-FN R(1):NEXT K
5Ø FOR J=1 TO S:FOR K=1 TO S-J+1
6Ø LET C(K,G,1)=FN R(1)+(C(K,F,1)+C
   (K+1,F,1))/2
7Ø LET C(K,G,2)=FN R(1)+H+(C(K,F,2)+
   C(K+1,F,2))/2
8Ø PLOT C(K,F,1),C(K,F,2):DRAW C(K+1,F,
   1)-PEEK 23677,C(K+1,F,2)-PEEK
   23678
9Ø DRAW C(K,G,1)-PEEK 23677,C(K,G,2)-
   PEEK 23678:DRAW C(K,F,1)-PEEK 23677,
   C(K,F,2)-PEEK 23678
1ØØ NEXT K:LET F=3-F:LET G=3-F:
    NEXT J
11Ø FOR y=4Ø TO Ø STEP -.75
12Ø PLOT Ø,y
13Ø FOR n=1 TO 1ØØ
14Ø LET a=RND*1Ø
15Ø LET b=5-RND*1Ø
16Ø IF a+PEEK 23677>255 THEN LET
    n=1ØØ:DRAW 255-PEEK 23677,b:GOTO
    19Ø
17Ø IF (PEEK 23678)+b<Ø THEN GOTO 15Ø
18Ø DRAW a,b
19Ø NEXT n
2ØØ NEXT y
21Ø FOR m=USR "a" TO USR "a"+7:READ
    a:POKE m,a:NEXT m
22Ø DATA 16,56,84,16,56,84,146,16
23Ø FOR n=1 TO 8Ø
24Ø PRINT AT 17+INT (RND*4),RND*31;
    BRIGHT 1;PAPER 4;INVERSE 1;CHR$ 144;
25Ø NEXT n
26Ø PRINT #1;INVERSE 1;AT Ø,4;
    "□RES'=";S;"□□□
    RUGGEDNESS=";RG;"□"
27Ø GOTO 27Ø
```

```
1Ø HIRES Ø,1:MULTI 5,13,Ø:COLOUR 6,14:
   X=1
15 BLOCK Ø,1ØØ,159,199,3:FOR Z=1 TO 8Ø:
   PLOT RND(1)*16Ø,RND(1)*1ØØ,2:NEXT Z
16 FOR Z=1 TO 2Ø:LINE Ø,1ØØ+XX,159,
   1ØØ+XX,9:X=X+.4:XX=XX+X:NEXT Z
2Ø DIM C(2ØØ,1,1):F=Ø:G=1:C(Ø,Ø,1)=
   15Ø:C(Ø,Ø,Ø)=1Ø
3Ø S=5Ø:L=14Ø/S:H=L/SQR(2)
4Ø FOR K=1 TO S:C(K,Ø,Ø)=C(Ø,Ø,Ø)+L*
   K-3+RND(1)*6
45 C(K,Ø,1)=C(K-1,Ø,1)-3+RND(1)*6:
   NEXT K
5Ø FOR J=1 TO S:FOR K=Ø TO S-J
6Ø C(K,G,Ø)=3-RND(1)*6+(C(K,F,Ø)+C
   (K+1,F,Ø))/2
7Ø C(K,G,1)=3-RND(1)*6-H+(C(K,F,
   1)+C(K+1,F,1))/2
8Ø LINE C(K,F,Ø),C(K,F,1),C(K+1,F,Ø),C
   (K+1,F,1),1
9Ø LINE C(K+1,F,Ø),C(K+1,F,1),C(K,G,Ø),C
   (K,G,1),2
95 LINE C(K,G,Ø),C(K,G,1),C(K,F,Ø),C(K,F,1),3
1ØØ NEXT K:F=1-F:G=1-F:NEXT J
11Ø GOTO 11Ø
```

```
1Ø GRAPHIC 1:COLOR 1,1,5,6
15 DRAW 2,Ø,7ØØ TO 1Ø23,7ØØ:
   PAINT 3,Ø,8ØØ
2Ø DIM C(1ØØ,1,1):F=Ø:G=1:C(Ø,Ø,1)=
   85Ø:C(Ø,Ø,Ø)=28
```

```
30 S=20:L=800/S:H=L/SQR(1)
40 FOR K=1 TO S:C(K,0,0)=C(0,0,0)+L*
   K-10+RND(1)*20
45 C(K,0,1)=C(K-1,0,1)-10+RND(1)*
   20:NEXT K
50 FOR J=1 TO S:FOR K=0 TO S-J
60 C(K,G,0)=20-RND(1)*40+(C(K,F,0)+
   C(K+1,F,0))/2
70 C(K,G,1)=20-RND(1)*40-H+(C(K,F,
   1)+C(K+1,F,1))/2
75 CL=(RND(1)*2)+2
80 DRAW CL,C(K,F,0),C(K,F,1) TO C(K+1,F,
   0),C(K+1,F,1)
90 DRAW CL TO C(K,G,0),C(K,G,1) TO C(K,F,
   0),C(K,F,1)
100 NEXT K:F=1-F:G=1-F:NEXT J
```

```
10 MODE1
20 DIMC(200,1,1):F=0:G=1:C(0,0,1)=
   150:C(0,0,0)=128
30 S=64:L=1024/S:H=L/SQR(2)
40 FORK=1TOS:C(K,0,0)=C(0,0,0)+L*K
   -5+RND(10):C(K,0,1)=C(K-1,0,1)
   -5+RND(10):NEXT
50 FORJ=1TOS:FORK=0TOS-J
60 C(K,G,0)=20-RND(40)+(C(K,F,0)+C
   (K+1,F,0))/2
70 C(K,G,1)=20-RND(40)+H+(C(K,F,1)
   +C(K+1,F,1))/2
80 MOVEC(K,F,0),C(K,F,1):DRAWC(K+1,F,
   0),C(K+1,F,1)
90 DRAWC(K,G,0),C(K,G,1):DRAWC(K,F,0),
   C(K,F,1)
100 NEXT:G=F:F=1-F:NEXT
```

```
10 PMODE4,1:PCLS: SCREEN1,1
20 DIMC(200, 1, 1):F=0: G=1: C(0,0,1)
   =150: C (0, 0, 0)=10
30 S=80:L= 230/S:H=L/ SQR (3):
   DEFFNR (X) =3 -RND (0) *6
40 FOR K=1TO S: C(K, 0, 0)=C (0, 0,
   0)+L*K- FNR (0): C(K,0,1)= C(K-1,
   0, 1)-FNR (0): NEXT
50 FORJ=1 TO S: FORK=0 TO S-J
60 C(K,G,0)= FNR (0)+ (C(K,F,0)
   +C(K+1, F, 0)) /2
70 C (K, G, 1)= FNR (0) -H+
   (C(K,F,1)+C (K+1, F, 1))/2
80 LINE (C(K, F, 0), C(K, F, 1))- (C(K+1,
   F, 0), C(K+1, F, 1)), PSET
90 LINE-(C (K,G,0), C(K, G, 1)), PSET: LINE
   - (C(K,F,0), C(K, F, 1)), PSET
100 NEXT:F=1-F: G=1-F: NEXT
110 GOTO110
```

The program draws small, irregular triangles, starting from the bottom left of the screen, and builds up the image as plotting continues across and up the screen.

Line 30 sets a size factor for the fractal triangles and specifies the length of one side and the height of the triangles. Line 40 loops 200 times, setting two array variables to the starting coordinates for each fractal triangle. Notice that there is a random factor, so the values will vary within a small range each time you RUN the program. Line 50 sets up two loops—one to move across the screen and draw triangles, and another to move up the screen.

The vertex of each triangle is specified at Lines 60 (the X coordinate) and 70 (the Y coordinate). Line 80 moves the cursor to the left-hand corner of the triangle, then draws the base. In the Spectrum program, notice the PEEKs subtracted from the coordinates to specify absolute points on the screen—without these, some points would lie off screen, and so cause an error when the computer tries to DRAW them. Line 90 DRAWs to the vertex of the triangle, then to the left corner—the start. Line 100 completes the first loop—to draw triangles across the screen—then sets variables to move the plotting position up screen and begin the next row of triangles. In the Spectrum version only, the rest of the program fills in the foreground to complete the picture.

MULTIPLE SHAPES

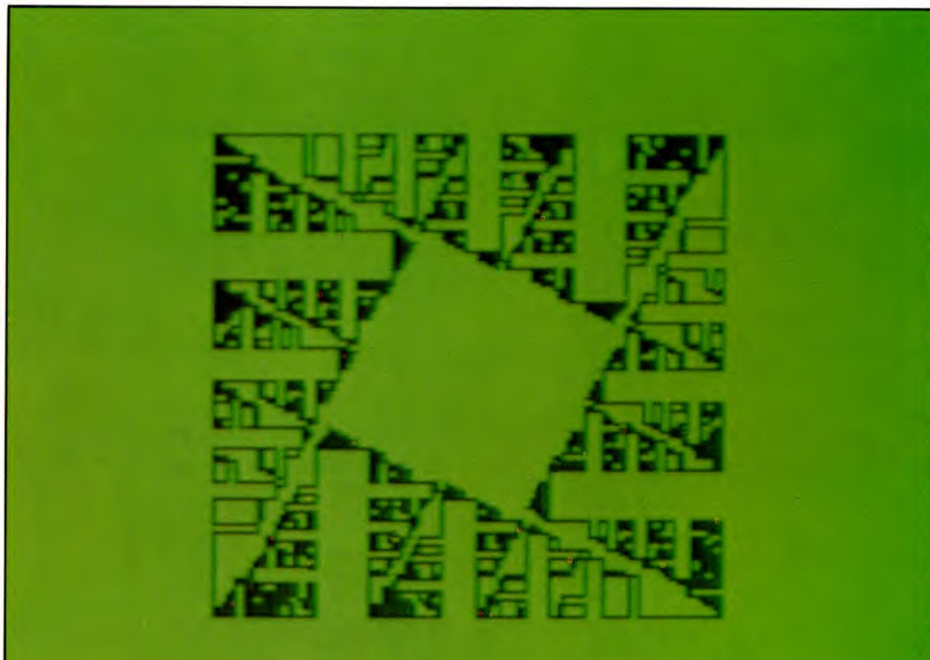
Changing the values of some of the variables in the programs you have seen so far will give different shapes. But the variation only affects the size, position and detail of the image, not its overall shape. The next program lets you specify any of a range of fractal elements, then see a fractal model built up from that particular shape. The large number of subroutines in this listing limits the number of levels of recursion and therefore the detail of the image drawn by the Commodores, so an alternative program which is suitable for the Commodore 64 is given later.

```
10 POKE 23658,8:LET F=0:LET A$="":LET
   CL=1
20 DIM X(100):DIM Y(100):DIM T(30):DIM
   U(30):DIM V(60):DIM W(60):DIM J(100)
25 BORDER 0:PAPER 7:INK 0:CLS
30 GOSUB 140
40 GOSUB 350
50 INPUT "No OF LEVELS OF RECURSION -
   □";NR:IF ABS (INT (NR))<1 THEN
   GOTO 50
60 LET F=1:LET N=0:CLS
70 PLOT INVERSE 1;OVER 1;INT(127+X(P)),
   INT(85+Y(P))
80 GOSUB 500:IF P>=2 THEN GOTO 80
90 LET A$=INKEY$:IF A$="" THEN GOTO
   90
100 PRINT #1;"Q TO QUIT, ANY OTHER TO
   CONTINUE"
110 LET A$=INKEY$:IF A$="" THEN GOTO
   110
120 IF A$<>"Q" THEN GOTO 25
130 CLS :STOP
140 IF F=0 THEN GOTO 170
150 PRINT "SAME INITIAL SHAPE [ Y/N ]
   ?□";
160 LET A$=INKEY$:IF A$<>"Y" AND
   A$<>"N" THEN GOTO 160
165 PRINT A$
170 IF F=0 OR A$="N" THEN GOSUB 230
180 FOR K=2 TO CV+1
190 LET P=K-1
200 LET X(P)=T(K):LET Y(P)=U(K)
210 NEXT K
220 RETURN
230 INPUT"NO OF VERTICES IN INIT'□";VT
240 FOR L=2 TO VT+1
250 INPUT "VERTEX□";(L-1);"="X,Y
260 LET T(L)=X*85: LET U(L)=y*85
270 IF L=2 THEN PLOT INT (127+T(L)),INT
   (85+U(L))
275 IF L<>2 THEN DRAW
   127+T(L)-PEEK 23677,85+U(L)-PEEK
   23678
280 NEXT L
290 PRINT "CLOSED CURVE [ Y/N ] ?□";
300 LET A$=INKEY$: IF A$<>"N" AND
   A$<>"Y" THEN GOTO 300
305 PRINT A$
310 IF A$="N" THEN LET CV=VT:PAUSE
   0:RETURN
320 LET CV=VT+1:LET T(CV+1)=T(2):
   LET U(CV+1)=U(2)
330 DRAW 127+T(CV)-PEEK 23677,85+U
   (CV)-PEEK 23678
340 RETURN
350 CLS :IF F=0 THEN GOTO 380
360 PRINT "SAME GENERATOR [ Y/N ] ?□";
370 LET A$=INKEY$:IF A$<>"Y" AND
   A$<>"N" THEN GOTO 370
375 PRINT A$
380 IF F=0 OR A$="N" THEN GOSUB 400
390 RETURN
400 INPUT "NO OF VERTICES IN
   GENERATOR NOT INCLUDING ENDS
   (0,0) AND (1,0)□";GN
420 PLOT INVERSE 1;OVER 1;85,85
430 FOR M=2 TO GN+1
440 INPUT "GENERATOR VERTEX□";
   (M-1); "="X,Y
450 IF ABS(INT (X))>1 OR ABS(INT (Y))>1
   THEN GOTO 440
460 LET V(M)=X:LET W(M)=Y:LETX=X
   *85+85:LET Y=85+Y*85:DRAW
   X-PEEK 23677,Y-PEEK 23678
470 NEXT M
480 DRAW 175-PEEK 23677,85-PEEK
```

```
   23678:PAUSE Ø
490 RETURN
500 IF NR=N THEN GOSUB 520
505 IF NR<>N THEN GOSUB 570
510 RETURN
520 FOR W=1 TO GN+1
530 LET P=P-1
540 IF ABS X(P)>127 OR ABS Y(P)>85
   THEN GOTO 560
550 DRAW 127+X(P)-PEEK 23677,
   85+Y(P)-PEEK 23678
560 NEXT W:RETURN
570 LET N=N+1
580 IF N=1 THEN LET JM=CV-1
585 IF N<>1 THEN LET JM=GN+1
590 FOR E=1 TO JM
595 IF P=1 THEN LET E=JM:NEXT
   E:RETURN
600 LET TX=X(P):LET TY=Y(P)
610 LET BX=X(P-1):LET BY=Y(P-1)
620 LET DX=TX-BX:LET DY=TY-BY
630 FOR F=2 TO GN+1
640 LET X(P)=DX*V(F)-DY*W(F)+BX
650 LET Y(P)=DY*V(F)+DX*W(F)+BY
660 LET P=P+1
670 NEXT F
680 LET X(P)=TX:LET Y(P)=TY
690 LET J(CL)=E:LET CL=CL+1:GOSUB
   500:LET CL=CL-1:LET E=J(CL)
700 NEXT E
710 LET N=N-1
720 RETURN
```

```
10 DIMX(5Ø),Y(5Ø),XT(1Ø),YT(1Ø),XG(2Ø)
   ,YG(2Ø),J(5Ø)
15 F=Ø:A$="":CL=Ø
20 MODE1:VDU28,Ø,2,39,Ø
30 GOSUB14Ø
40 GOSUB35Ø
50 INPUT"NO OF LEVELS OF
   RECURSION□";NR%:IF NR%<1THEN5Ø
60 F=1:N=Ø:CLG
70 MOVE 64Ø+X(P),48Ø+Y(P)
80 GOSUB5ØØ:IFP>ØTHEN8Ø
90 VDU7
100 CLS:PRINT"Q TO QUIT, ANY OTHER KEY
   TO CONTINUE"
110 A$=GET$
120 IF A$="Q" THEN CLS:END ELSE 2Ø
140 IFF=ØTHENGOSUB23Ø:GOTO18Ø
150 INPUT"SAME INITIAL SHAPE (Y/N)?";A$
170 IF A$="N" THEN GOSUB23Ø
180 FORK=1TOCV
190 P=K-1
200 X(P)=XT(K):Y(P)=YT(K)
210 NEXT
220 RETURN
230 INPUT"NO OF VERTICES IN INITIAL
   SHAPE□";VT%:IFVT%<1THEN23Ø
240 FORL=1TOVT%
250 PRINT"VERTEX□";L;:INPUT" □-□"
   X,Y:IF ABS(X)>1ORABS(Y)>1THEN25Ø
260 XT(L)=X*512:YT(L)=Y*42Ø
270 IFL=1THENMOVE64Ø+XT(L),48Ø+YT
   (L)
280 DRAW64Ø+XT(L),48Ø+YT(L):NEXT
290 INPUT"CLOSED CURVE (Y/N)?";A$
310 IFA$="N"THENCV=VT%:DEL=
   INKEY(2ØØ): RETURN
320 CV=VT%+1:XT(CV)=XT(1):YT(CV)
   =YT(1)
330 DRAW64Ø+XT(CV),48Ø+YT(CV)
340 RETURN
350 CLS: IFF=Ø THEN 38Ø
360 INPUT"SAME GENERATOR (Y/N)?";A$
380 IF A$="N" OR F=Ø THEN GOSUB 4ØØ
390 RETURN
400 PRINT"NO OF VERTICES IN
   GENERATOR"
410 INPUT"NOT INCLUDING ENDS (Ø,Ø)
   AND (1,Ø)";GN%:IFGN%<1 THEN41Ø
420 CLG:MOVE64Ø,48Ø
430 FORM=1TOGN%
440 PRINT"GENERATOR VERTEX□";M;:
   INPUT"□-□"X,Y
450 IF ABS(X)>1OR ABS(Y)>1 THEN 44Ø
460 XG(M)=X:YG(M)=Y:X=X*512+64Ø:
   Y=Y*42Ø+48Ø:DRAWX,Y
470 NEXT
480 DRAW1152,48Ø
490RETURN
500 IF(NR%=N)GOSUB52ØELSE GOSUB 57Ø
510 RETURN
520 FORL=1TOGN%+1
530 P=P-1
550 DRAW64Ø+X(P),48Ø+Y(P)
560 NEXT:RETURN
570 N=N+1
580 IFN=1THENJM=CV-1 ELSE
   JM=GN%+1
590 FORJ=1TOJM
600 TX=X(P):TY=Y(P)
610 BX=X(P-1):BY=Y(P-1)
620 DX=TX-BX:DY=TY-BY
630 FORE=1TOGN%
640 X(P)=DX*XG(E)-DY*YG(E)+BX
650 Y(P)=DY*XG(E)+DX*YG(E)+BY
660 P=P+1
670 NEXT
680 X(P)=TX:Y(P)=TY
690 J(CL)=J:CL=CL+1:GOSUB5ØØ:
   CL=CL-1:J=J(CL)
700 NEXT
710 N=N-1
720 RETURN
```

```
10 DIMX(5Ø),Y(5Ø),XT(1Ø),YT(1Ø),XG(2Ø),
   YG(2Ø),J(5Ø)
20 PMODE4,1:COLORØ,1:PCLS:CLS
30 GOSUB14Ø
40 GOSUB35Ø
50 INPUT"NO. OF LEVELS OF
   RECURSION -";NR:NR=INT(NR):IF
   NR<1 THEN 5Ø
60 F=1:N=Ø:PCLS:SCREEN1,Ø
70 LINE-(127+X(P),96-Y(P)),PRESET
80 GOSUB5ØØ:IF P>Ø THEN 8Ø
90 A$=INKEY$:IF A$="" THEN 9Ø
100 CLS:PRINT"Q TO QUIT, ANY OTHER
   KEY TO□□□□□CONTINUE"
110 A$=INKEY$:IF A$="" THEN 11Ø
120 IF A$<>"Q" THEN 2Ø
130 CLS:END
140 IF F=Ø THEN 17Ø
150 PRINT"SAME INITIAL SHAPE (Y/N) ?";
160 A$=INKEY$:IF A$<>"Y" AND
   A$<>"N" THEN 16Ø ELSEPRINTA$
170 IF F=Ø OR A$="N" GOSUB 23Ø
180 FORK=1TOCV
190 P=K-1
200 X(P)=XT(K):Y(P)=YT(K)
210 NEXT
220 RETURN
230 INPUT"NO. OF VERTICES IN INITIAL
   SHAPE□";VT:VT=INT(VT):IF VT<1
   THEN 23Ø
240 FORL=1 TO VT
250 PRINT"VERTEX□";L;:INPUT"□-□";
   X,Y:IF ABS(X)>1 OR ABS(Y)>1 THEN
   25Ø
260 XT(L)=X*95:YT(L)=Y*95
270 IF L=1 THENLINE-(127+XT(L),96-
   YT(L)),PRESET ELSE LINE-(127+XT(L),
   96-YT(L)),PSET
280 NEXT
290 PRINT"CLOSED CURVE (Y/N) ?";
300 A$=INKEY$:IF A$<>"N" AND
   A$<>"Y" THEN 3ØØ ELSEPRINTA$
310 IF A$="N" THEN CV=VT:GOSUB73Ø:
   RETURN
320 CV=VT+1:XT(CV)=XT(1):YT(CV)=YT
   (1)
330 LINE-(127+XT(CV),96-YT(CV)),PSET:
   GOSUB73Ø
340 RETURN
350 PCLS:IF F=Ø THEN 38Ø
360 PRINT"SAME GENERATOR (Y/N) ?";
370 A$=INKEY$:IF A$<>"Y" AND
   A$<>"N" THEN 37Ø ELSEPRINTA$
380 IF F=Ø OR A$="N" GOSUB 4ØØ
390 RETURN
400 PRINT"NO. OF VERTICES IN
   GENERATOR"
410 INPUT"NOT INCLUDING ENDS (Ø,Ø)
   AND□□□□(1,Ø)□-□";GN:GN=
   INT(GN):IF GN<1 THEN 41Ø
420 DRAW"BM8Ø,96"
430 FOR M=1 TO GN
440 PRINT"GENERATOR VERTEX";M;:INPUT
   "□-□";X,Y
450 IF ABS(X)>1 OR ABS(Y)>1 THEN 44Ø
```



An example of the shape generator in action

```
460 XG(M)=X:YG(M)=Y:X=X*95+80:Y=
    96-Y*95:LINE-(X,Y),PSET
470 NEXT
480 LINE-(175,96),PSET:GOSUB730
490 RETURN
500 IF NR=N GOSUB520 ELSEGOSUB570
510 RETURN
520 FORL=1TOGN+1
530 P=P-1
540 IF ABS(X(P))>127 OR ABS(Y(P))>95
    THEN 560
550 LINE-(127+X(P),96-Y(P)),PSET
560 NEXT:RETURN
570 N=N+1
580 IF N=1 THEN JM=CV-1 ELSE
    JM=GN+1
590 FORJ=1TOJM
600 TX=X(P):TY=Y(P)
610 BX=X(P-1):BY=Y(P-1)
620 DX=TX-BX:DY=TY-BY
630 FORE=1TOGN
640 X(P)=DX*XG(E)-DY*YG(E)+BX
650 Y(P)=DY*XG(E)+DX*YG(E)+BY
660 P=P+1
670 NEXT
680 X(P)=TX:Y(P)=TY
690 J(CL)=J:CL=CL+1:GOSUB500:CL=
    CL-1:J=J(CL)
700 NEXT J
710 N=N-1
720 RETURN
730 A$=INKEY$:SCREEN1,0:K=1000
740 K=K-1:IF K>0 AND INKEY$=""
    THEN 740
750 RETURN
```

When you RUN the program, Line 23Ø asks you how many corners (vertices) you want in the shape that forms the starting point for the fractal. It's best to draw the shape out first on a sheet of paper. Mark two dots representing the start and end of the line then link these with a number of short straight lines. Count up the number of corners and enter this into the program. But remember that you will then have to specify coordinates (Line 25Ø) for all the corners, so keep the number small—three or four, say. The coordinates should be given values between Ø and 1. The loop between Lines 24Ø and 28Ø lets you enter coordinates and draw the initial shape. You can then choose whether the figure you have specified should be closed or left with an opening (Line 29Ø).

The next input stage lets you specify the shape with which each straight line will be replaced—this shape is usually called the generator. Draw this out and enter the information as you did for the initial shape—the subroutine that inputs and draws the shape lies between Lines 4ØØ and 49Ø. Now you are asked to specify the number of levels of recursion. When you have entered the value, Line 8Ø calls a subroutine to determine whether the program is being run for the first time (in which case it branches to the main routine—Lines 57Ø to 72Ø—to draw the fractals), or whether it should recycle and give you the option to redefine the generator.

TESTING

As an example, enter a value of 3 for the number of vertices in the initial shape. Then enter coordinates −.5, −.2 for vertex 1; Ø, Ø.4 for vertex 2; Ø.5, −Ø.2 for vertex 3. If now you reply N to the prompt 'CLOSED CURVE ?', a triangle without the base drawn will appear on the screen. Similarly, if you enter 3 for the number of vertices in the generator, you could specify coordinates: Ø.2, Ø for vertex 1; Ø.4, Ø.8 for vertex 2; Ø.6, Ø for vertex 3. This gives a shape of a base-less triangle sitting on a line. Now enter about 5 for the level of recursion, and see what fractal is drawn.

```
10 PRINT "♥"
20 CX=160:CY=100:IT=0
30 X=.50001:Y=0
40 GOSUB 330
50 HIRES 0,1
60 GOSUB 110
70 FOR I=1 TO 10:GOSUB 250:
   NEXT I
80 GOSUB 380
90 GOSUB 250
100 GOTO 80
110 T=Y
120 S=SQR((X*X)+(Y*Y))
130 Y=SQR((-X+S)/2)
140 X=SQR((X+S)/2)
150 IF T<0 THEN X=-X
160 RETURN
170 S=(AX*AX)+(AY*AY)
180 AX=6*(AX/S)
190 AY=-6*(AY/S)
200 RETURN
210 TX=X: TY=Y
220 X=(TX*AX)-(TY*AY)
230 Y=(TX*AY)+(TY*AX)
240 RETURN
250 GOSUB 210
260 X=1-X
270 GOSUB 110
280 IF RND(1)<.5 THEN X=-X
290 IF RND(1)<.5 THEN Y=-Y
300 X=1-X
310 X=X/2:Y=Y/2
320 RETURN
330 PRINT "ENTER X & Y FACTORS":
    INPUT AX,AY
340 PRINT "ENTER SCALE FACTOR":
    INPUT OP
350 GOSUB 170
360 SC=2*CX/OP
370 RETURN
380 PLOT SC*(X-.5)+CX,CY-SC*Y,1
390 RETURN
```

CLIFFHANGER: A SAD DEMISE

Death comes to all of us—but to none more often than Willie. He expires five times every game, but he still lives to climb the cliff again another day!

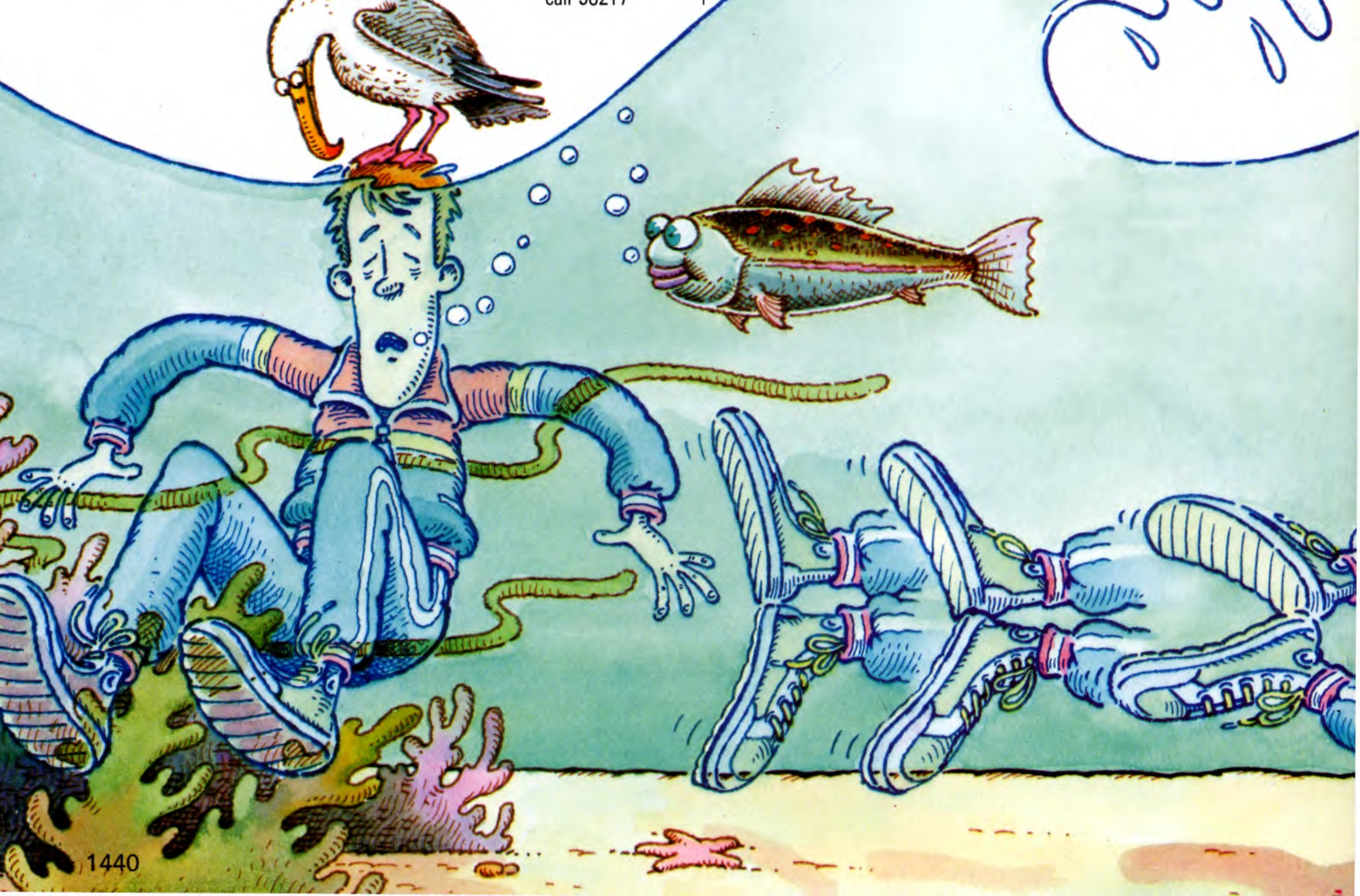
However good you are at playing games the grim reaper is bound to get his hands on Willie sooner or later. But don't be too upset. He does have five lives—not as many as a cat but a good many more than poor normal mortals. When he is dead, you have to bury him, reset all the variables and check whether the game is over.

On the Commodore and Acorn computers you are also getting some sound effects. The sounds you can get from the Spectrum are much simpler and are supplied in the programs as you go along. And though the Dragon and Tandy can produce fairly sophisticated sounds they have to be produced using processor time. On the Commodore and Acorn there are separate chips which can produce sounds while the main processor is doing something else—but on the Dragon and Tandy the main processor has to be used, so you cannot make a sound and play the game at the same time.

S

The following routine kills off Willie and checks to see whether he can be resurrected.

```
        org 59652
die     ld de,196
        ld hl,1086
        call 949
        ld de,131
        ld hl,1646
        call 949
dead    ld hl,(57332)
        ld bc,15616
        ld a,45
        call 58217
        ld de,32
        add hl,de
        ld (57332),hl
        ld bc,57000
        ld a,40
        ld de,258
        call 58970
        ld de,30
        ld hl,878
        ld a,r
        ld l,a
        call 949
        ld hl,(57332)
        ld de,704
        sbc hl,de
        jr c,dead
        ld a,9
        ld (60005),a
        ld a,(57343)
        dec a
        ld (57343),a
        jp nz,58606
        ld hl,330
        ld a,142
        ld b,11
        ld ix,57992
        call 58155
        call 58939
        ld de,261
```



- FOR WHOM THE BELL TOLLS
- DROPPING DOWN THE SCREEN
- DECREMENTING LIVES
- PRINTING UP SCORE
- ENDING THE GAME


```
ld hl,1646
call 949
ld de,392
ld hl,1Ø86
call 949
ld de,261
```

```
ld hl,1646
call 949
ld a,5Ø
ld (58734),a
jp 58576
```

Once Willie has fallen down a hole, been bitten by snakes, hit by a boulder or drowned in the sea, the first thing that happens is that his death knell is sounded. In fact, two bells are tolled to give the correct sombre sound. This is done by calling the **BEEPER** routine at 943 twice and feeding different parameters to it each time through the HL and DE registers. For details of how the pitch and duration of the bell tolls are worked out see page 732.

DESCENT INTO THE INFERNO

When Willie is dead he descends to the inferno that lies just off the bottom of the screen. And to do that you start off by blanking out his head—you don't want a string of decapitated heads left down the screen!

So Willie's screen position—which points to the position of his head—is loaded up from its storage location 57,332 into the HL register. BC is loaded with the address of blank sky at the top of the screen. A is set to cyan on cyan. And the **print** routine at 58,217 is called which prints a piece of sky over Willie's head.

Then 32 is added to the screen pointer in HL, via the DE register. This moves it one line down the screen. The result is stored back in 57,332 to update that pointer, too.

BC is loaded up with the start address of the picture of Willie with his legs together. A is set to Willie's colours, blue on cyan. DE is loaded with 258, specifying a block of one by two—the width of the block is specified by the contents of the D register and the depth is specified by the contents of the E register $1 \times 256 + 2 = 258$. And the block print routine, **blk**, at 58,97Ø is called. This prints a picture of Willie with his legs together one square down the screen from the position of his last appearance.

And while Willie descends, a celestial siren sounds. Again the DE and HL register pairs are loaded with parameters to be passed into the **BEEPER** routine when it is called at 949. But this time, instead of a constant tone, the program needs a sort of warbling sound. This is done by loading the contents of the *refresh* register, R, into the low byte of the HL register which is just plain L. It's done via the accumulator because there is no **ld l,r** command.

The refresh register is used to refresh dynamic memory. This is a hardware facility that you needn't bother with here. The important thing about the refresh register is that it has a different value in it each time you look at it. So each time this part of the routine is performed it will produce a different pitch.

SIX FOOT UNDER

The position of Willie's head is loaded into HL from 57,332 again and 7Ø4 is loaded into DE and subtracted from the contents of HL. 7Ø4 is 32×22, so it marks the beginning of the second to last line. As Willie is two character squares high, this subtraction sets the carry flag until Willie's feet have touched the bottom of the screen.

While the carry flag is still set, **jr c,dead** takes the processor back to the beginning of the burial routine again. Willie will be buried one character square deeper and more celestial sirens will wail—at a different pitch. But when Willie's feet touch the floor of the upper world, he has been buried deep enough. The carry flag will no longer be set by the subtraction and the jump will no longer be made. The processor will proceed with the next instruction.

With Willie in Hades it's time to play a happy little sound. So A is loaded with 9 and the **sound** routine at 6Ø,ØØ6 is called. This plays a short version of the tune. For details of how the music routine works see Cliffhanger on page 966.

Then the number of lives have to be updated—or rather, *down*dated. Willie has lost a life. So the number of lives is loaded up into the accumulator from 57,343, decremented and stored back in 57,343.

If there are still some lives left, the **jp nz** instruction sends the processor back to the initialization program at 58,6Ø6. If not, the processor proceeds into the game-over routine.

CHARNEL NUMBER FIVE

When Willie has been killed five times, the game is over and you need to print 'GAME OVER!!' This is done by calling the message, or **me**, routine at 58,155. But as always parameters have to be fed through to it in the HL, A, B and IX registers.

The print position is carried in HL. The colour is set in A. B carries the message length and IX is used as the data pointer. The message data is at 57,892.

The routine that prints up the score at 58,939 is then called to print up the final score.

Next, the chimes that signal the end of the game sound. These are produced by calling the BEEPER routine at 949 three times with different values in HL and DE each time to produce different tones.

The initial delay value is then reset by loading the accumulator with 5Ø and loading it into memory location 58,732. Then the processor jumps back into the game at 58,578 and sets it up for you to try again if that's what you fancy.

On the Commodore, checking to see whether Willie is dead is a relatively easy matter. You simply have to check for sprite collisions. This routine does that—and takes the appropriate action to bury poor Willie six foot under.

```
      ORG 2568Ø
      LDA $DØ1E
      STA $Ø35C
      LDA $DØ1F
      STA $Ø35D
      LDA $Ø35C
      AND #64
      BEQ AA
      LDA #4
      STA $Ø384
      JSR $685Ø
      INC $CØØØ
      LDA #155
      RTS
AA    LDA $Ø35C
      AND #1
      BEQ BB
      LDA #3
      STA $Ø384
      JSR $685Ø
DD    DEC $CØØ1
      LDA #254
      RTS
BB    LDA $Ø35D
      AND #1
      BEQ CC
      LDA #3
      STA $Ø384
      JSR $685Ø
      JMP DD
CC    LDA #Ø
      RTS
```

Memory location $DØ1E is the sprite-to-sprite collision detection register. Each sprite has a bit in this register and whenever a sprite is in collision with another sprite the appropriate bit in this register is set.

And memory location $DØ1F is the sprite-to-UDG collision detection register. Whenever a sprite is involved in a collision with a UDG, the appropriate bit is set.

THINGS THAT GO BUMP

The contents of the sprite-to-sprite collision detection register are loaded up into the accumulator and stored in $Ø35C, which is a convenient location in the cassette buffer for temporary storage. Then the contents of the sprite-to-UDG collision detection register are loaded into the accumulator and stored in $Ø35D.

When these two registers are read by loading their contents into a register, they are automatically cleared, ready to detect the next collision. So if a particular register has to be referred to more than once, you have to store the contents in a convenient location.

REWARDING COLLISION

Not all the collisions Willie makes are unpleasant. Sometimes he bumps into a reward.

The rewards are in sprite six. So the contents of $Ø35C are loaded into the accumulator and ANDed with 64. This isolates bit six. If it is not set—and Willie has not yet reached his reward—the BEQ instruction branches forward over the next part of the routine.

But if bit six is set, the branch is not made and the processor goes to the next instruction. The accumulator is loaded with 4 which is then stored in memory location $Ø384. This is the location that is used to pass parameters into the sound effects routine at $685Ø. You will see this listing in a moment. A 4 in $Ø384 makes that routine play the winning sound, and when the processor jumps to the subroutine at $685Ø it does just that.

When it comes back it moves up a level by incrementing the level number which is stored in memory location $CØØØ. Then the accumulator is loaded with 155 and the processor hits the RTS and returns.

This routine uses the accumulator to carry parameters out of the routine. These depend on the nature of the collision and tell other routines what to do next.

COLLISION COURSES

The contents of $Ø35C are loaded into the accumulator again and ANDed with 1. Willie is on the zero sprite, so if he has collided with any other sprite—a snake, a boulder, a cloud or a gull—the zero bit of the register will be set.

If it is not set, the BEQ branches forward. But if it is, the processor performs the next little routine.

A 3 is loaded into the accumulator and stored in $Ø384, and the processor jumps to the subroutine at $685Ø. This plays the losing sound.

And when the processor returns the number of lives in $CØØ1 are decremented. The accumulator is then loaded with 254 which it carries out of the routine when the processor hits the RTS and returns.

DATA DETECTION

But Willie is not just killed off by accidents with sprites. Bumping into a UDG—like the sea or a hole—can be just as dangerous. So the contents of memory location $Ø35Ø are loaded up into the accumulator and ANDed with 1. Again, any collision involving Willie will set bit zero of that register because Willie is on sprite zero.

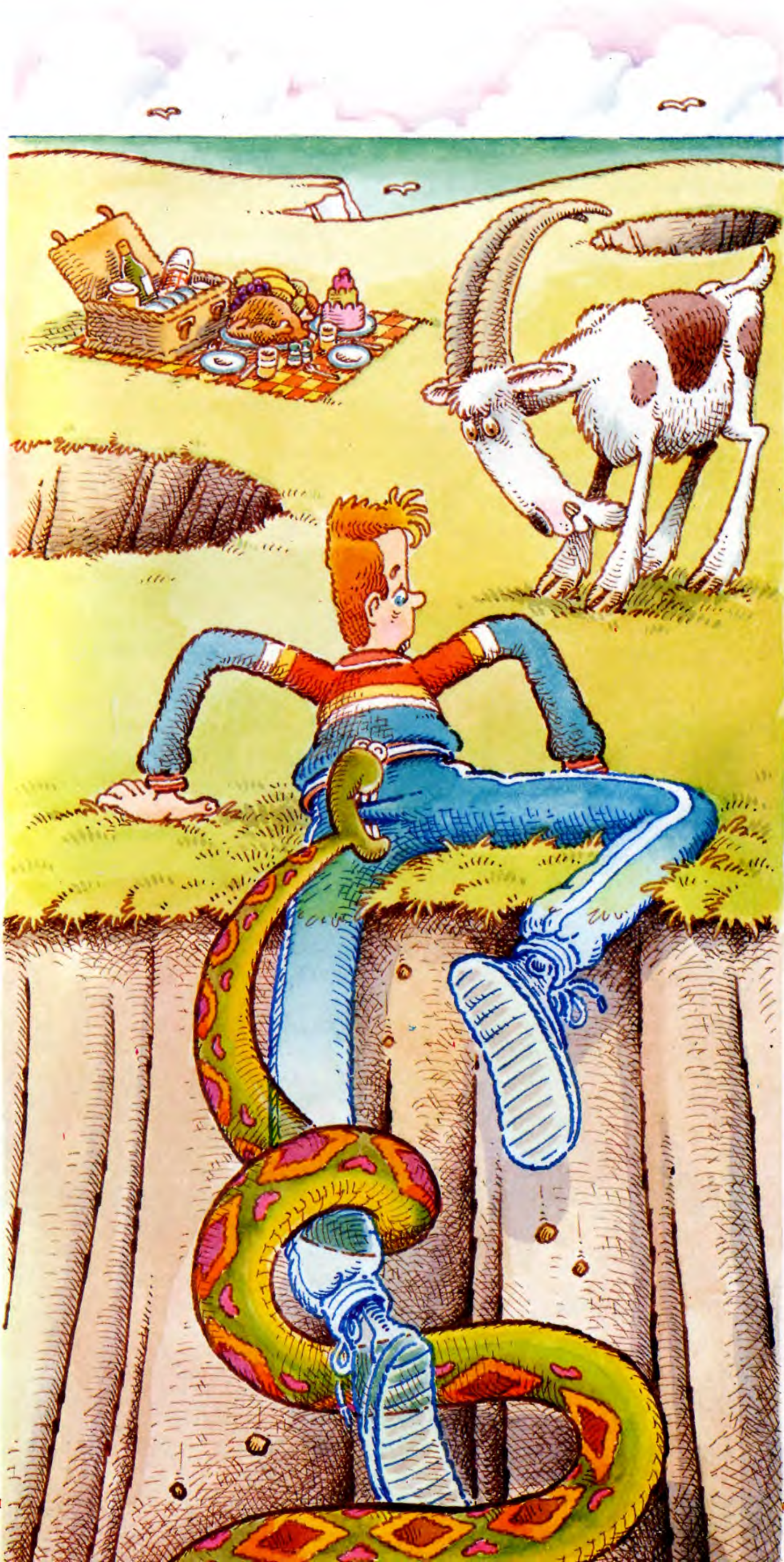
If this bit is not set either, Willie has bumped into neither a sprite nor a UDG and the BEQ instruction branches the processor forward to the label CC. There, the accumulator is loaded with Ø before returning.

But if this bit is set, Willie has bumped into something bad again and is dead. So 3 is loaded into the accumulator and stored in $Ø384 and the processor jumps to the routine at $685Ø again to play the losing sound. Then the processor jumps back to DD, where it decrements the number of lives and exits the routine with 254 in the accumulator again. Whatever bad thing Willie hits—be it sprite or UDG—the result is the same.

NOISES OFF

The routine above called the sound effects routine at $685Ø several times. This is it:

```
      ORG 26704
      LDA #15
      STA $D418
      LDA #30
      STA $D401
      LDA #0
      STA $D406
      LDA $0384
      CMP #1
      BNE AA
      LDA #1
      STA $D405
      JMP FF
AA    LDA $0384
      CMP #2
      BNE BB
      LDA #85
      STA $D405
FF    LDA #0
      STA $D404
      LDA #33
      STA $D404
RET   RTS
BB    LDX #30
      LDA #0
      STA $D404
      LDA #129
      STA $D404
      LDA #240
      STA $D406
GG    LDA $0384
      CMP #3
      BEQ CC
      INX
      CPX #50
      BNE DD
EE    LDA #0
      STA $D404
      RTS
CC    DEX
      CPX #10
      BEQ EE
DD    STX $D401
      LDA #90
LOOPA LDY #255
LOOP  DEY
      BNE LOOP
      CLC
      SBC #1
      BNE LOOPA
      JMP GG
```



Making a noise is not an easy job. First of all, you have to decide how loud it is going to be, what the pitch is and how long the sound goes on. Then you have to define its shape—how quickly it reaches its maximum volume, how soon the peak is over, how long it is sustained and how quickly it finally dies away.

These vital parameters of any sound are called its envelope—see pages 1138–1144.

LICKING THE ENVELOPE

A is loaded with 15 which is stored in $D418. This is the register on the SID chip that selects the filters—bits four to seven—and sets the volume—bits zero to three. A 15 selects the maximum volume and leaves the filters alone.

The number 30 is then put into $D01 which controls the high byte of the frequency. And 0 is put into D406, which is the envelope generator that controls the sustain and release.

The parameter that is going to tell the routine what sound to make is then loaded up from $0384. It is compared to 1.

If a 1 is not found, the BNE instruction branches the processor forward to the label AA. But if it is found, a 'blip' is required and the process continues.

A 1 is put into $D405. This is the register which controls the attack and decay of the note. The attack is specified by bits four to seven and is set to its minimum value 0. The decay—in bits zero to three—is set to 1, giving the characteristic 'blip' sound.

The processor then jumps forward to the label FF where the sound is output.

BEEPING ABOUT

If no 'blip' is required the process skips to the label AA where the contents of $Ø384 are loaded into the accumulator again. They are then compared to 2—a 2 means that a 'beep' is required.

Again if it is not found the BNE instruction branches the processor forward over the beep routine. Otherwise, the processor continues.

A is loaded with 85, which is stored in $D4Ø5. This sets both the attack and the decay to 5.

SOUND OUT

Whether a blip or a bleep is required, the processor then ends up at FF, where a Ø is loaded into the accumulator and stored in $D4Ø4. This is the byte that controls the output of sound and setting it to Ø starts the release—in other words, it turns the last sound to be made off.

A 33 is then loaded up into the accumulator and stored in $D4Ø4. This sets bit five—which gives a sawtooth waveform—and bit zero which switches on the attack and decay. In other words, it switches the sound of the next note on.

The blip or beep begins and the processor hits an RTS and returns.

LOONY TUNES

The next part of the routine plays the winning sound or the losing sound. They are in fact the same sound, but the winning sound is played with the pitch rising triumphantly and the losing sound is played with the pitch dropping dismally.

In both cases, the pitch starts off from the same place and 3Ø is loaded into the index register X to start the count.

A Ø is loaded into the accumulator and stored in $D4Ø4 to turn the last note off again. Then 129 is put into $D4Ø4 to turn on white noise.

The number 24Ø is put into $D4Ø6 to set the sustain to the maximum 15. Then the routine looks to see which sound it should play.

The contents of $Ø384 are loaded into the accumulator and compared with 3. A 3 in $Ø384 means that the losing sound is to be played and the BEQ instruction branches the processor forward.

But if the winning sound is to be played, the branch is not made and the processor proceeds to the next instruction. INX increments the contents of the X register and they are compared to 5Ø.

If the result is not 5Ø, the BNE instruction sends the processor forward. But if it is 5Ø, the processor loads its accumulator with Ø and stores it in $D4Ø4, turning the last note off. It then returns.

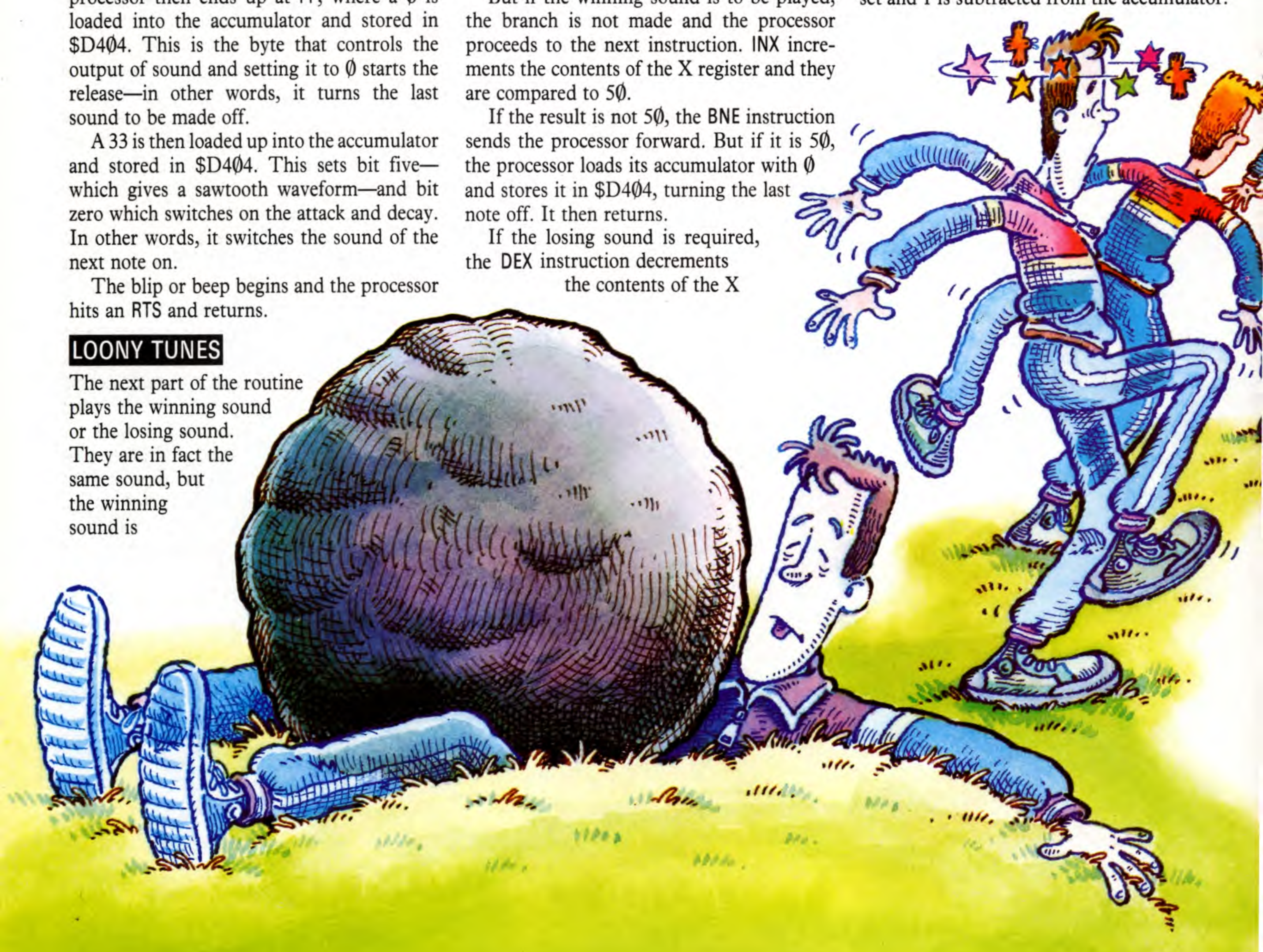
If the losing sound is required, the DEX instruction decrements the contents of the X register and they are compared to 1Ø. If 1Ø is found, the BEQ branches the processor back to the label EE, where the note is turned off and the routine is exited again.

If not, the processor continues. The resulting contents of the X register are stored in $D4Ø1—the high byte of the pitch—whether X has been incremented or decremented.

PROLONGING THE AGONY

A is loaded with 9Ø and Y is loaded with 255. These numbers are going to be used as the parameters for two nested loops which are here simply to slow the game down at this point to give each note time to be heard.

The contents of the Y register are decremented and the BNE sends the processor round and around that LOOP until Y has counted down to zero. Then the carry flag is set and 1 is subtracted from the accumulator.



Remember, you normally set the carry flag before a subtraction to prevent an extra borrow being taken into account. So here, 2 is really subtracted from the contents of the accumulator.

And if A has not counted down to zero the BNE instruction sends the processor back round LOOPA to load Y up with 255 and start decrementing that all over again.

When A has finally counted down to zero, the processor jumps back to give the next 'note'.

The following routine does the rest of the sound effects for the BBC version of Cliffhanger and makes the necessary funeral arrangements if Willie is dead.

Don't forget to set the computer up as usual before you key it in.

```
30 FORPASS=ØTO3STEP3
40 RESTORE
8Ø DATA1,Ø,4,Ø,14Ø,Ø,2Ø,Ø
9ØFORA%=&14DETO&14E5:READ?A%:NEXT
1ØØ P%=&14E7
11Ø [OPTPASS
12Ø .Bell
13Ø JSR&152B
14Ø LDA#8
15Ø LDX#&91
16Ø LDY#&14
17Ø JSR&FFF1
18Ø LDA#7
19Ø LDX#&DE
2ØØ LDY#&14
21Ø JSR&FFF1
22Ø RTS
23Ø ]
27Ø DATA1,2,255,Ø,Ø,255,Ø,Ø,126,255,255,
    255,126,126
28Ø FORA%=&14FDTO&15ØA:READ?A%:
    NEXT
29Ø DATA1,Ø,1,Ø,21Ø,Ø,55,Ø
3ØØ FORA%=&15ØCTO&1513:READ?A%:
    NEXT
31Ø P%=&1515
32Ø [OPTPASS
33Ø .Cry
34Ø JSR&152B
35Ø LDA#8
36Ø LDX#&FD
37Ø LDY#&14
38Ø JSR&FFF1
39Ø LDA#7
4ØØ LDX#&C
41Ø LDY#&15
42Ø JSR&FFF1
43Ø RTS
44Ø ]
48Ø DATA 1,Ø,241,255,1Ø,Ø,1,Ø
49Ø FORA%=&1BCØTO&1BC7:READ?A%:
    NEXT
5ØØ P%=&1BC8
51Ø [OPTPASS
52Ø .Plod
53Ø JSR&152B
54Ø LDA#7
55Ø LDX#&CØ
56Ø LDY#&1B
57Ø JSR&FFF1
58Ø RTS
59Ø ]
63Ø DATA 1,129,1,Ø,-1,2Ø,1Ø,2Ø,126,Ø,Ø,
    -126,126,126
64Ø FORA%=&1BD5TO&1BE2:READ?A%:
    NEXT
65Ø DATA17,Ø,1,Ø,128,Ø,1Ø,Ø
66Ø FORA%=&1BE3TO&1BEA:READ?A%:
    NEXT
67Ø P%=&1BEB
68Ø [OPTPASS
69Ø .Jump
7ØØ JSR&152B
71Ø LDA#8
72Ø LDX#&D5
73Ø LDY#&1B
74Ø JSR&FFF1
75Ø LDA#7
76Ø LDX#&E3
77Ø LDY#&1B
78Ø JSR&FFF1
79Ø RTS
8ØØ ]
81Ø $&2ØD6=CHR17+CHR$131+CHR$
    17+CHR$4+CHR$31+CHR$5+CHR$
    2Ø+"□□□□□□□□□□□"
82Ø $&2ØE9=CHR$31+CHR$5+
    CHR$21+"□Game Over□"
83Ø $&2ØF8=CHR$31+CHR$5+
    CHR$22+"□□□□□□□□□□
    □"
85Ø P%=&21Ø7
86Ø [OPTPASS
87Ø .Dead
88Ø LDA&7B
89Ø BEQLb1
9ØØ RTS
91Ø .Lb1
92Ø DEC&89
93Ø LDA&7D
94Ø ORA#&8Ø
95Ø STA&7D
96Ø LDA#15
97Ø LDX#Ø
98Ø JSR&FFF4
99Ø LDA&89
1ØØØ BEQLb2
1Ø1Ø RTS
1Ø2Ø .Lb2
1Ø3Ø LDX#Ø
1Ø4Ø .Lb3
1Ø5Ø LDA&2ØD6,X
1Ø6Ø JSR&FFEE
1Ø7Ø INX
1Ø8Ø CPX#&49
1Ø9Ø BNELb3
11ØØ RTS
111Ø ]NEXT
```

FOUR SOUNDS

The first part of this program is made up of four modules that work in exactly the same way. Each starts off with a block of DATA which defines the sound effect in question. This is READ into a data table in memory so that the machine code program can access it.

Then, when the processor passes into the assembly language program it jumps to the subroutine at &152B. This is the one that allows you to turn the sound effects off (see page 1243). Next, the A register is loaded with 8, and the X and Y registers are loaded with the low and high bytes of the appropriate piece of envelope data. The processor then jumps to the subroutine at &FFF1. This is an OSWORD call and the 8 in A means that it defines a sound envelope with the data provided in the fourteen memory locations from the address given in X and Y onwards. Next, A is loaded with 7. This means that the sound itself will be output when the OSWORD call is made. Again, X and Y are used to carry the low and high bytes of the base address of the data for the sound which is given here. The sound itself only requires eight bytes to define it. Then &FFF1 which actually makes the sound is called. When the sound has been made, the processor returns.

The sounds here are a bell, a cry, Willie's walking sound and Willie's jumping sound. You'll notice that the data for the envelope for the cry and the jump are given within the program here. The envelope for the bell is the same as the one used for the crunch which was defined before on page 1243. And Willie's walking sound does not need to have its envelope defined because a simple plodding sound does not need an envelope. The computer is told that it does not need an envelope by the second and third byte of the sound data. These are the equivalent of the second parameter of a BASIC sound command—the two bytes of data are the low byte and high byte of the parameter respectively.

With the basic SOUND command (see page

233), a positive number in the second parameter position is the number of the envelope the sound is to use. And a negative number is a volume as no envelope is to be used. Here the 241, 255 in DATA are the equivalent of −15.

TO DIE, TO SLEEP

The routine that deals with the death of Willie starts in Line 81Ø. The first three lines POKE string data into a data table to print up 'Game Over' if it is required. The CHR$17s are the equivalent of a COLOUR command, so the first four CHR$ commands define a foreground and a background colour. The CHR$31 moves the cursor to the X and Y positions following. So three lines on the screen—2Ø, 21 and 22—have spaces, the words 'Game Over' and more spaces are printed up starting at X position 5. The spaces are printed to give a background panel around the words.

The assembly language program begins with LDA&7B in Line 83Ø. This loads the accumulator with Willie's Y coordinate which is stored in &7B. So far, if Willie has hit upon a hazard that has killed him, he is precipitated down to the bottom of the screen. If Y is zero he has reached it and he is well and truly dead.

So the BEQ instruction in Line 89Ø branches the processor on into the main death routine, if Willie's Y coordinate is zero. If it is not, there is still a chance that Willie might be alive, so the processor does not make the branch and returns.

If Willie is definitely dead, the first thing to be done is to decrement his number of lives. The DEC&89 in Line 92Ø does that. The control byte in &7D is then loaded up into the accumulator and ORed with &9Ø. This sets bit seven which is the flag that orders up the next screen. The result is stored back in &7D.

A is loaded with 15, X is loaded with Ø and the processor jumps to the subroutine at &FFF4. This is equivalent to a *FX15,Ø which clears all sounds.

The number of lives in &89 is then loaded up into the accumulator. If Willie has no lives left, the BEQ instruction branches the processor over the RTS, into the 'Game Over' routine. But if Willie is still blessed with another incarnation or two, the processor returns to rewind his mortal coil.

THE GAME IS OVER

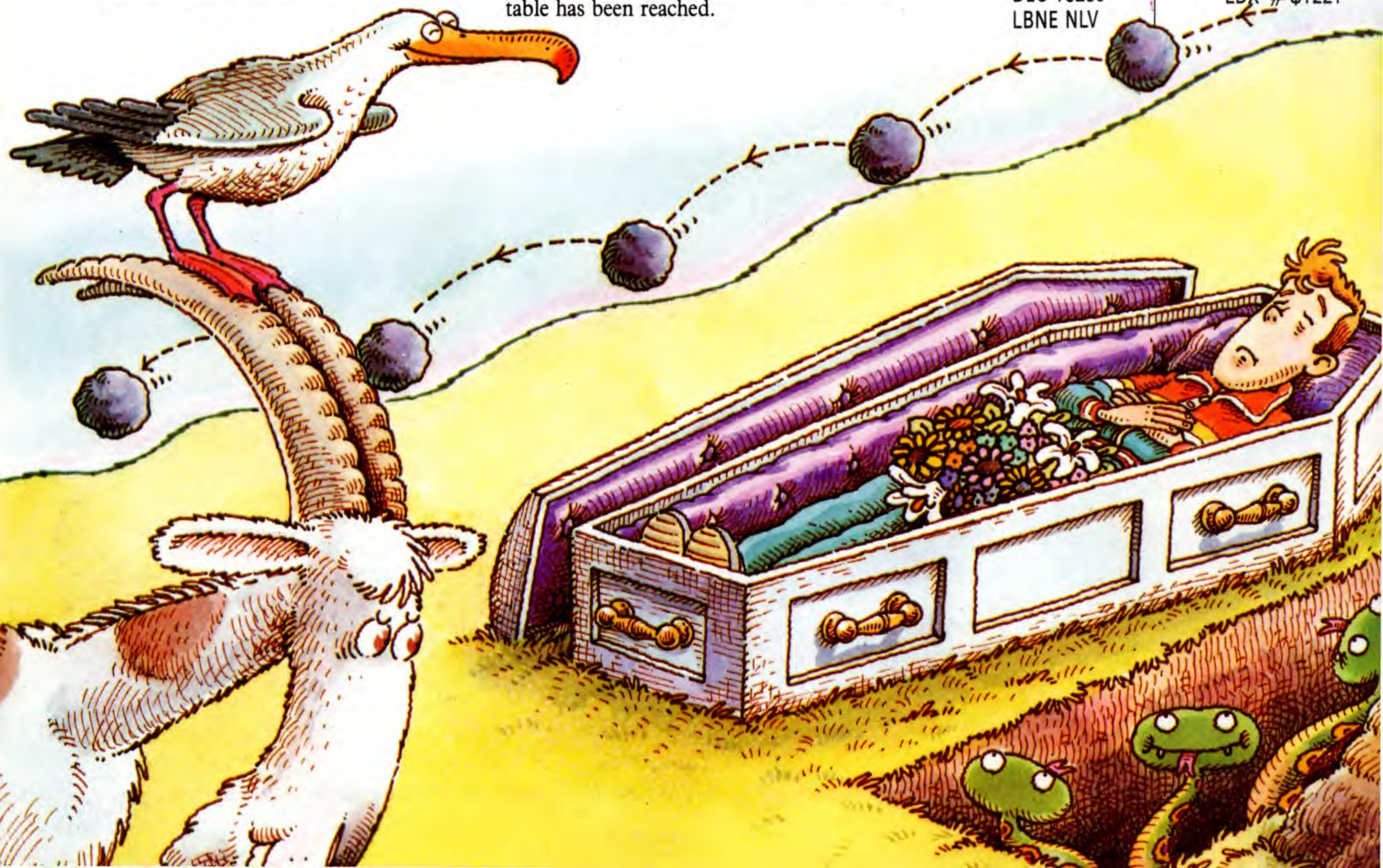
The number Ø is loaded into the X register which is going to be used as an index to count along a data table.

Then the appropriate byte of the data table constructed in Lines 81Ø to 83Ø is loaded up into A and output to the screen by the subroutine at &FFEE. X is incremented and compared with &49 to see if the end of the table has been reached.

If it hasn't, the processor branches back to pick up and output the next byte. If it has, the processor proceeds, hits the RTS in Line 11ØØ and returns.

The following routine puts Willie in his grave:

```
ORG  2Ø56Ø
DIE  LDA #136
     LDX #14Ø
     JSR SOUND
     LDA #131
     LDX #213
     JSR SOUND
DI   LDX 18249
     LDU #1536
     JSR CHARPR
     LEAX 254,X
     STX 18249
     LDU #17774
     JSR CHARPR
     LEAX 254,X
     JSR CHARPR
     LDA #3Ø
     LDX #113
     JSR SOUND
     LDX 18249
     CMPX #6912
     BLO DI
     DEC 18239
     LBNE NLV
PB   LDA #5
PAA  LDX #65535
     LEAX −1,X
     BNE PAA
     DECA
     BNE PB
     JSR CLS
     LDA $FF22
     ANDA #15
     STA $FF22
     STA $FFC2
     STA $FFC4
     STA $FFC6
     LDY #$5ØB
     LDX #$7Ø1
     STX ,Y++
     LDX #$DØ5
     STX ,Y++
     LDX #$2ØØF
     STX ,Y++
     LDX #$16Ø5
     STX ,Y++
     LDX #$1221
```



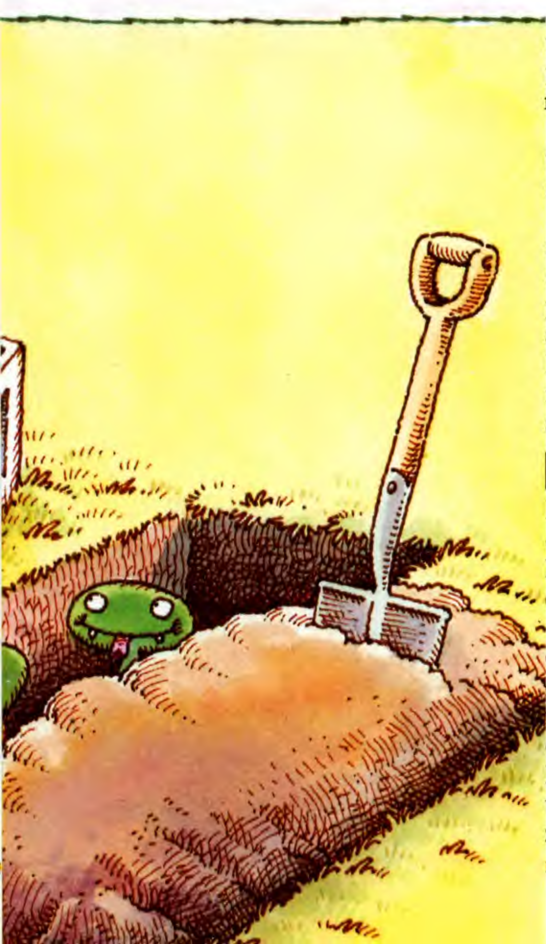
```
STX ,Y++
LDA #2ØØ
LDX #255
JSR SOUND
LDA #2ØØ
LDX #2ØØ
JSR SOUND
LDA #255
LDX #255
JSR SOUND
LDA #1ØØ
STA DLL+1
LBRA GBIN
SOUND    EQU $5133
CHARPR   EQU $4BCA
DLL      EQU $51ED
GBIN     EQU $4BE2
NLV      EQU $4BF7
CLS      EQU $4ACC
```

THE KNELL

The routine starts by sounding a two-note death knell for dear departed Willie. But the actual sounding is done by the SOUND routine which will be given in the next part of Cliffhanger.

Before you attempt to test this program, don't forget to put an RTS at the start address of SOUND, $5133. Otherwise, the program will crash.

SOUND requires two parameters to be fed into it to specify the pitch and duration of the sound. These are taken into the routine by the A and X registers. It is plain to see that the two notes produced by the same SOUND routine are going to be very different in this case as the two parameters are so different each time it's called.



GOING DOWN

As Willie drops down the screen into the inferno below, the bits left behind have to be overprinted with sky.

So X is loaded with the contents of 18,249, which point to Willie's screen position. And U is loaded with 1536, the address of the sky at the top of the screen.

The CHARPR routine is then called to print a block of sky over Willie's top half.

X is loaded with X plus 254. And this is stored back in 18,249 to move Willie's pointer down the screen one character square.

U is then loaded with 17,774, which is the start address of the data for the picture of Willie with his legs together. So when the processor jumps to the CHARPR subroutine, it prints up the top half of Willie one character square down the screen from where it was printed last time.

X is incremented by 254 and CHARPR is called again to print Willie's bottom half.

While Willie is going down the screen he lets out a death cry. This is done by loading up A and X again and jumping to SOUND.

When Willie dies he descends all the way to the bottom of the screen. So you need to check whether he has got there. This is done by loading X with the contents of 18,249 and comparing them with 6912, the start of the 28th line of the screen.

If Willie's screen position in X is lower than 6912, the BLO instruction sends the processor round to start the DI loop again and moves Willie down one more character square.

But if Willie's screen position is not lower than 6912, he has reached the bottom of the screen and the processor continues.

In that case, the contents of 18,239—which stores the number of lives Willie has left—is decremented and the LBNE NLV sends the processor back to the routine that gives Willie a new life and sets the last screen up again. A long branch is used here because the NLV routine was given several parts of Cliffhanger ago and is certainly more than 128 bytes away.

GAME OVER

But if the contents of 18,239 have been decremented to zero, Willie has no more lives left, the game is over and the processor continues.

A is loaded with 5 and X is loaded with 65,535. X is then decremented and the processor loops back to decrement it again, unless it has counted down to zero. If it has, A is decremented and the processor jumps back to load X up again with 65,535. The process is then repeated until A has counted down to zero as well.

The processor goes around this loop 5 × 65,535 times. This gives the player pause for thought over Willie's final, tragic demise.

When the processor finally drops out of the delay loop, it jumps off to the CLS subroutine. This clears the screen.

Next the contents of $FF22 are loaded up and ANDed with 15. The result is stored in $FF22, $FFC2, $FFC4 and $FFC6.

This tells the VDG (video display generator) and SAM (synchronous address multiplexer) chips that you are going back into text mode and is the reverse of what you did when you changed into graphics mode earlier in the program (see page 1Ø42).

Y is loaded with $5ØB—this is the position you are going to print 'GAME OVER!' in.

X is loaded with $7Ø1, the screen codes for the letters GA. These are stored on the screen in the position pointed to by Y, and Y is incremented twice.

Then X is loaded with the screen codes for ME and those are stored in the new position pointed to by Y, two places to the right of the start of GA. Then □O, VE and R! are loaded up and printed in the subsequent places across the screen.

Next the 'game over' sounds are played. It is three notes. So A and X are loaded up and the SOUND routine is called three times.

Then the number 1ØØ is loaded into the accumulator and stored in memory location $51EE. This sets the delay back to its starting value, so next time the game is played it will start at the same initial speed.

Finally, the processor makes a long branch back to the beginning of the program at GBIN.

MUSIC WHILE YOU WORK

Most of the best games you can buy have impressive sound effects or music for added entertainment. Here is a program to let your micro soothe you with music while you type in a program or develop one of your own.

Most games programs have a musical introduction or a short refrain to mark certain events, such as when you score points or lose a life in a game. If, however, you have played some of the popular games that have tunes on the Commodore 64 or the Acorn computers the most impressive effect is that the tunes are played not just momentarily, but throughout the game. The reason these micros can run a program and play music at the same time is that they have a sound-generating circuit that works independently of the central processor, unlike other micros, such as the Spectrum, Dragon and Tandy, which use the central processor to process sound.

There are a few games available for the Spectrum which manage to play music constantly throughout the program. But this is done by *very* careful timing so that the two things appear to happen simultaneously. This is not practical in a general-purpose program like those given here, so only Commodore and Acorn versions are listed.

The programs work by interrupts, so at regular intervals a certain event is detected, causing the program to branch to a routine to play the tune. But in case you want a moment's quiet—for example, when the telephone rings—Acorn users can disable the interrupts by a single keypress, then turn them on again by pressing another key. On the Commodore 64 where it is not possible to do this, it is a simple matter to turn down the volume control on the TV set.

Enter the program now, but notice that it contains a section of machine code (given in DATA statements), so save a copy to disk or tape before you RUN.

```
1000 S=0:FOR N=49152 TO 49407:
   READ A:POKE N,A:S=S+A:NEXT N
1050 IF S<>5666 THEN 1400
1100 S=0:FOR N=24576 TO 24631:
   READ A:POKE N,A:S=S+A:NEXT N
1150 IF S<>6270 THEN 1500
1200 S=0:FOR N=28672 TO 28778:
   READ A:POKE N,A:S=S+A:NEXT N
1250 IF S<>12659 THEN 1600
1300 PRINT "♥":SYS 24576:END
1400 PRINT "CHECK LINES 2000–
   2060":END
1500 PRINT "CHECK LINES 2100–
   3020":END
1800 PRINT "CHECK LINES 4000–
   4050":END
2000 DATA 37,17,37,3,37,17,27,3,47,16,7,3,
   37,17,7,3,63,19,62,8
2010 DATA 37,17,7,3,63,19,17,3,37,17,7,3
2020 DATA 63,19,7,3,37,17,17,3,47,16,7,3,
   110,15,72,8
2030 DATA 63,19,37,3,63,19,27,3,42,18,7,3
2040 DATA 63,19,7,3,154,21,62,8
2050 DATA 63,19,7,3,154,21,17,3,63,19,7,3
2060 DATA 154,21,7,3,63,19,17,3,42,18,7,3,
   37,17,72,8
2100 DATA 37,17,37,3,37,17,27,3,47,16,7,3,
   37,17,7,3,63,19,62,8
2110 DATA 37,17,7,3,63,19,17,3,37,17,7,3
2120 DATA 63,19,7,3,37,17,17,3,114,11,7,3,
   63,19,72,8
2130 DATA 216,12,7,3,107,14,7,3,70,15,7,3,
   37,17,7,3
2140 DATA 154,21,27,3,63,19,7,3,63,19,7,3,
   37,17,42,8
2150 DATA 47,16,7,3,37,17,7,3,63,19,37,3,
   154,21,37,3,227,22,72,8
2160 DATA 0,0,1,1,0,0,1,1,0,0,1,1,0,0,1,1
2170 DATA 0,0,1,1,0,0,1,1,0,0,1,1,0,0,1,1
2180 DATA 0,0,1,1,0,0,1,1,0,0,1,1,0,0,1,1
2190 DATA 0,0,1,1,0,0,1,1,0,0,1,1
3000 DATA 169,0,162,25,157,0,212,202,208,
   250,169,15,141,24,212,169,1,141
3010 DATA 1,195,169,0,141,2,195,141,4,195,
   169,32,141,3,195,169,246,141
3020 DATA 5,195,169,64,141,5,212,120,169,
   0,141,20,3,169,112,141,21,3,88,96
4000 DATA 174,4,195,173,13,220,41,1,240,3,
   32,16,112,76,49,234,173,2,195
4010 DATA 240,17,206,0,195,208,11,169,0,
   141,2,195,173,3,195,141,4,212
4020 DATA 96,206,1,195,240,13,169,1,205,1,
   195,208,5,169,0,141,6,212,96
4030 DATA 189,0,192,141,0,212,232,189,0,
   192,141,1,212,232,189,0,192,141
4040 DATA 0,195,232,189,0,192,141,1,195,
   232,142,4,195,173,5,195,141,6,212
1050 DATA 172,3,195,200,140,4,212,169,1,
   141,2,195,96,0
```

When you RUN the BASIC program, it sums the numbers in the DATA statements, and if you have made an error a prompt will direct you to the lines you should check.

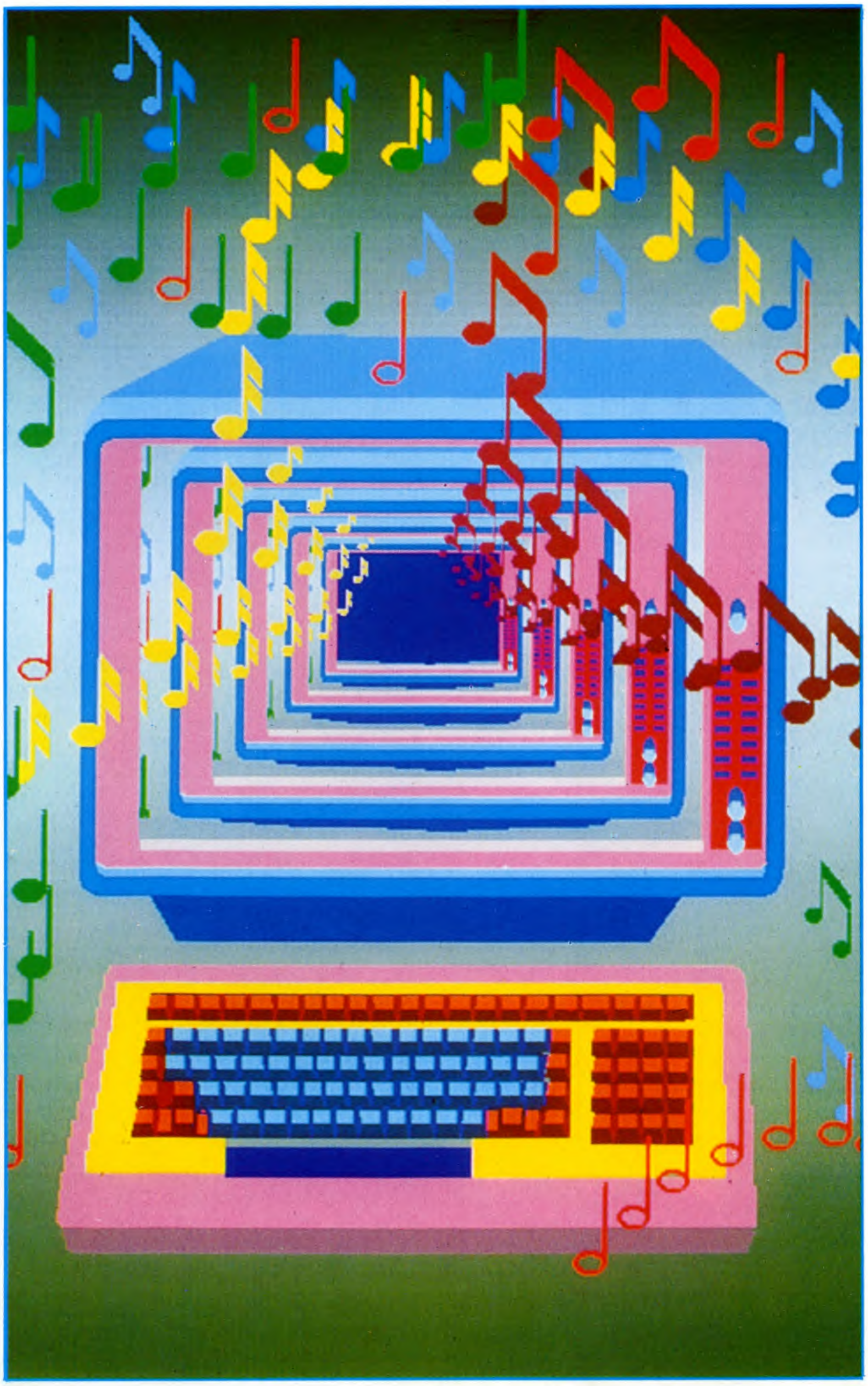
The data is read and checked as three blocks, so it is important that you enter it as listed, otherwise the check fails. The tune is planned so that it repeats in a loop without overlaps or gaps. This requires careful timing of the notes, in relation to the interrupt cycle. Furthermore, the data for the tune is encoded, so it no longer looks like the usual values you would enter to play a tune. All this means that there is little scope for you to change the tune, unless you are familiar with the techniques of programming by interrupts.

Once the program has been RUN, the machine code is called automatically to start the music. You can then NEW to get rid of the BASIC and the tune will be unaffected. To stop the tune, press RUN/STOP and RESTORE, and to restart it type SYS 24576.

```
10 M%=&900
11 !M%=&FFF80002
12 M%!4=&00040000
13 M%!8=0
20 X%=-1
30 REPEAT
40 READ W%
50 X%=X%+1
60 ?(X%+M%+10)=W%
70 UNTIL W%=255
80 FOR PA%=0 TO 3 STEP 3
90 P%=M%+X%+11
100 [ OPT PA%
140 .L% INC M%+9
150 LDA M%+9
160 CMP #10
170 BNE E%
180 LDA #0
190 STA M%+9
200 LDY M%+8
210 LDA M%+10,Y
220 STA M%+4
230 LDA #7
240 LDX #M% MOD 256
```

If you enjoy the soothing strains of background music while you work, you will like this simple, interrupt-driven machine-code music program which won't exhaust the memory

- SIMULTANEOUS PLAYING AND PROCESSING
- MACHINE CODE PROGRAMMING
- USING THE INTERRUPTS
- THE SOUND CIRCUITRY



```
250 LDY #M% DIV 256
260 JSR &FFF1
270 INC M%+8
280 LDY M%+8
290 LDA M%+1Ø,Y
300 CMP #255
31Ø BNE E%
32Ø LDA #Ø
33Ø STA M%+8
34Ø .E% RTS
35Ø ]
36Ø NEXT PA%
37Ø CALL L%
38Ø *KEY 8 ?&22Ø=L% MOD 256:
    ?&221=L% DIV 256:*FX14,4IM
39Ø *KEY 9 *FX13,4IM
4ØØ DATA 165,161,165,161,165,145,157,149,
    137,69,89,1Ø1,117,137
41Ø DATA 145,69,85,117,133,145,149,69,89,
    117,165,161
42Ø DATA 165;161,165,145,157,149,137,69,
    89,1Ø1,117,137
43Ø DATA 145,69,85,117,149,145,137,69,89
44Ø DATA 145,149,157,165,81,1Ø1,129,169,
    165,157,81,97,121,165,157
45Ø DATA 149,69,89,117,157,149,145,96,117,
    117,165,117
46Ø DATA 165,165,213,161,165,161,165
47Ø DATA 161,165,161,165,145,157,149,137,
    69,89,1Ø1,117,137
48Ø DATA 145,69,85,117,133,145,149,69,89,
    117,165,161
49Ø DATA 165,161,165,145,157,149,137,69,
    89,1Ø1,117,137
5ØØ DATA 145,69,85,117,149,145,137,69,89,255
```

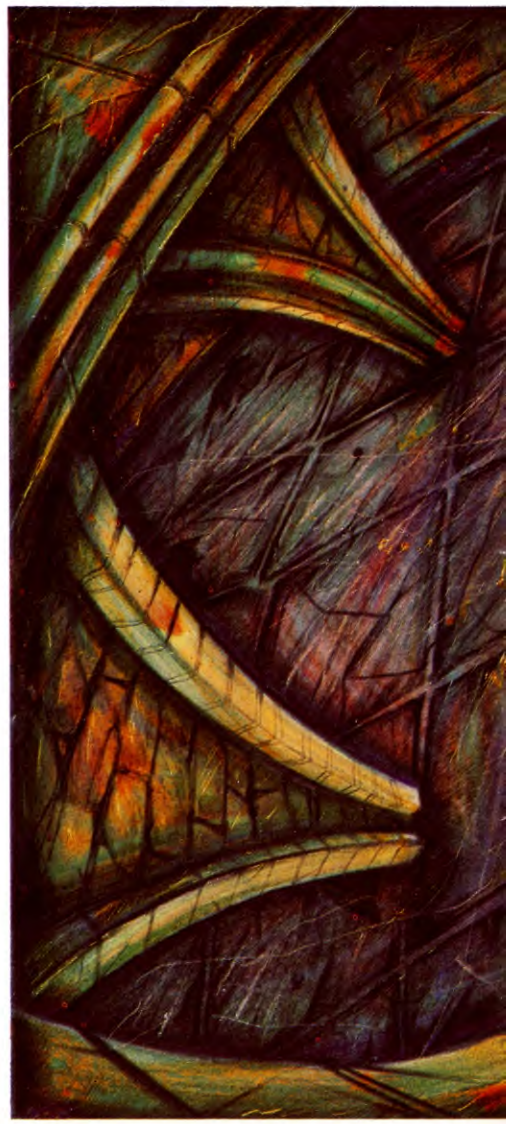
When you RUN the program, it gives a listing of the assembly code. If you don't wish the listing to appear, change the two 3s in Line 8Ø to 2s. To start the tune, press [f8], then you can NEW and use the micro in the usual way, with the tune playing. You can stop the tune by pressing either [f9] or [BREAK]—in either case, [f8] restarts it.

Lines 38Ø and 39Ø set the key definitions to start and stop the tune. If you wish to set these functions to any of the other user-defined keys, then merely substitute the numbers at these lines. After you've saved the listing, you might wish to play about with the data to see the effect, but notice that timing is crucial.

ESCAPE: BUILDING UP THE ADVENTURE

These program lines are part two of Escape, following on directly from part one. Don't forget to SAVE this ready for part three.

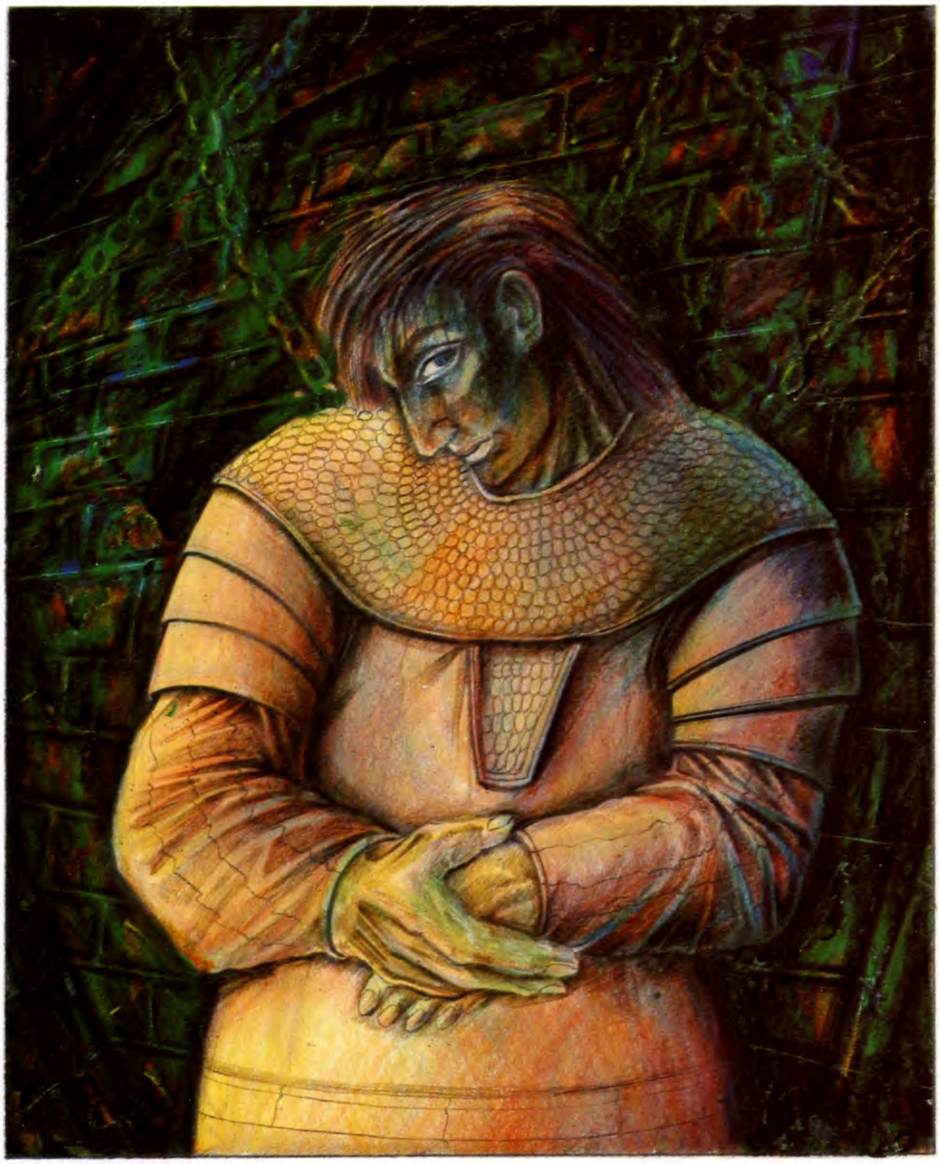
```
20 PRINT "LOADING TEXT AND DECODER":
   LOAD ""CODE
30 LET S$="□□□□□□□□□": LET
   Z$="": FOR Z=1 TO 32: LET
   Z$=Z$+S$: NEXT Z
40 LOAD "" DATA A(): LOAD "" DATA Z()
180 LET PP=0: LET BB=1: LET V=10: LET
   DW=1: LET D$="": LET M$="": LET
   J$="": LET U$="": LET X=0: LET Q
   =0: LET JK=0: LET QQ=0: LET OP=0
190 LET SS=1: LET C=0: LET M=0: LET
   XX=0: LET J=INT (RND*18)+1: LET
   G=INT (RND*18)+1: LET TT=1: LET
   II=1: LET VV=0: LET F=0: LET
   KK=INT (RND*21)+1: LET NN=70:
   GOSUB 4500: DIM M$(17): LET M$=S$
820 IF L=10 AND N$="W" AND OP=1
   THEN GOSUB 3810: GOTO 270
830 IF L=10 AND N$="W" AND OP>0
   THEN LET NN=60: GOSUB 3960: PAUSE
   200: GOTO 270
840 IF L=3 AND N$="D" AND D=1 THEN
   GOSUB 1760
850 IF L=10 AND N$="W" THEN LET
   OP=1: GOSUB 3810: GOTO 270
860 IF N$="N" AND N=1 THEN LET
   L=L-3: GOTO 270
870 IF N$="S" AND S=1 THEN LET
   L=L+3: GOTO 270
880 IF N$="E" AND E=1 THEN LET
   L=L+1: GOTO 270
890 IF N$="W" AND W=1 THEN LET
   L=L-1: GOTO 270
900 IF N$="U" AND U=1 THEN LET
   L=L-6: GOTO 270
910 IF N$="D" AND D=1 THEN LET
   L=L+6: GOTO 270
920 IF L=16 AND N$="D" AND
   K(7)<>-1 THEN LET NN=29: GOSUB
   3960
930 PRINT "YOU CAN'T DO THAT HERE-"'
   "THINK AGAIN!": PAUSE 100: GOTO 270
940 LET N=0: LET S=0: LET E=0: LET
   W=1: LET U=0: LET D=0
950 CLS : LET NN=2: GOSUB 3960
960 IF BB=1 THEN LET NN=3: GOSUB
   3960: PRINT ' FLASH 1;" PRESS ANY KEY
   TO BEGIN ": PAUSE 0: LET BB=0
970 RETURN
980 CLS : LET NN=4: GOSUB 3960: LET
   N=0: LET S=0: LET E=0: LET W=1:
   LET U=0: LET D=0: RETURN
990 IF INT (RND*18)=6 THEN GOSUB 2550
1000 IF INT (RND*18)=12 AND DW=1
   THEN GOSUB 2740: RETURN
1010 CLS : LET NN=5: GOSUB 3960: LET
   N=1: LET S=0: LET E=0: LET W=0
   LET U=0: LET D=0: RETURN
1020 IF INT (RND*18)=10 AND DW=1
   THEN GOSUB 2740: RETURN
1030 IF INT (RND*18)=1 THEN CLS :
   GOSUB 2550
1040 CLS : LET NN=6: GOSUB 3960: LET
   N=0: LET S=1: LET E=1 LET W=1:
   LET U=1: LET D=1: IF OP=1 THEN LET
   NN=62: GOSUB 3960
1050 RETURN
1060 IF INT (RND*18)=17 AND DW=1
   THEN GOSUB 2740: RETURN
1070 CLS : LET NN=7: GOSUB 3960: LET
   N=0: LET S=1: LET E=0: LET W=1:
   LET U=1: LET D=0: RETURN
1080 CLS : LET NN=8: GOSUB 3960: LET
   N=0: LET S=1: LET E=1: LET W=1:
   LET F=0: LET U=0: LET D=0
1090 IF E$(L,1)<>CHR$ 32 THEN LET
   NN=9: GOSUB 3960: PAUSE 50
1100 IF E$(L,1)<>CHR$ 32 AND J<8
   THEN LET NN=10: GOSUB 3960
1110 IF E$(L,1)<>CHR$ 32 AND J>7
   THEN LET NN=11: GOSUB 3960: LET
   F=1: LET QQ=QQ+1: GOTO 1420
1120 RETURN
1130 IF INT (RND*18)=16 THEN GOSUB
   2550
1140 CLS : LET NN=28: GOSUB 3960: LET
   N=1: LET S=0: LET E=1: LET W=0:
   LET U=0: LET D=0: RETURN
1150 LET NN=13: GOSUB 3960: LET N=1:
   LET S=0: LET E=1: LET W=0: LET
   U=0: LET D=0: RETURN
1160 IF INT (RND*18)=14 AND DW=1
   THEN GOSUB 2740: RETURN
1170 CLS : LET NN=14: GOSUB 3960:
   PAUSE 150: LET NN=15: GOSUB 3960:
   LET N=0: LET S=1: LET E=1: LET
   W=0: LET U=0: LET D=0: RETURN
1180 IF INT (RND*18)=17 THEN GOSUB
   2550
1190 LET N=1: LET S=0: LET E=1: LET
   W=1: LET F=0: LET U=0: LET D=0
1200 IF G<=4 THEN LET E$(L)=""
1210 CLS : LET NN=16: GOSUB 3960
1220 IF E$(L,1)<>CHR$ 32 THEN PRINT
   '"HE STIRS": LET F=1
1230 RETURN
1240 IF INT (RND*18)=5 THEN GOSUB 2550
1250 CLS
1260 LET N=1: LET S=0: LET E=0: LET
```



Continue entering Escape, *INPUT*'s new adventure game. LOAD in the existing program and add these lines. The program cannot be RUN until it is completed


```
   W=1: LET U=0: LET D=0
1270 LET NN=17: GOSUB 3960
1280 RETURN
1290 LET N=1: LET S=0: LET W=1: LET
   E=1: LET F=0: LET U=0: LET D=0
1300 LET NN=18: GOSUB 3960
1310 IF K(19)=-1 THEN LET F=0: LET
   NN=19: GOSUB 3960
1320 RETURN
1330 IF INT (RND*18)=1 AND DW=1 THEN
   GOSUB 2740: RETURN
1340 CLS
1350 LET N=0: LET S=1: LET E=1: LET
   W=0: LET U=0: LET D=0
1360 LET NN=20: GOSUB 3960
1370 RETURN
1380 CLS
1390 LET N=0: LET S=1: LET E=1: LET
   W=1: LET U=0: LET D=0
1400 LET NN=21: GOSUB 3960
1410 IF E$(L,1)<>CHR$ 32 THEN LET
   NN=22: GOSUB 3960: LET F=1: LET
   QQ=QQ+1
1420 IF QQ>4 THEN PRINT "SHE
   SCREAMS.": PAUSE 100: PRINT "TWO
   GUARDS APPEAR.": PAUSE 100: PRINT
   "YOU SURRENDER.": PAUSE 100: STOP
1430 RETURN
1440 LET N=1: LET S=0: LET E=0: LET
   W=1: LET F=0: LET U=0: LET D=0
1450 LET NN=23: GOSUB 3960
1460 LET NN=24: GOSUB 3960: LET F=1
1470 IF INT (RND*18)<3 THEN LET F=0:
   LET NN=63: GOSUB 3960
1480 RETURN
1490 IF INT (RND*18)=3 AND DW=1 THEN
```

```
   GOSUB 274Ø: RETURN
15ØØ CLS
151Ø LET N=Ø: LET S=1: LET E=Ø: LET
   W=1: LET U=Ø: LET D=Ø
152Ø LET NN=25: GOSUB 396Ø
153Ø IF K(17)=-1 THEN LET NN=26:
   GOSUB 396Ø: LET D=1
154Ø RETURN
155Ø CLS
156Ø LET N=1: LET S=Ø: LET E=1: LET
   W=Ø: LET U=Ø: LET D=Ø
157Ø LET NN=27: GOSUB 396Ø
158Ø RETURN
159Ø CLS
16ØØ LET N=Ø: LET S=1: LET E=Ø: LET
   W=Ø: LET U=1: LET D=Ø
161Ø LET NN=12: GOSUB 396Ø
162Ø IF K(7)=-1 THEN LET NN=64:
   GOSUB 396Ø: LET D=1
163Ø RETURN
164Ø IF INT (RND*18)=1 THEN GOSUB 255Ø
165Ø CLS : LET N=1: LET S=1: LET E=1:
   LET W=Ø: LET U=Ø: LET D=1
166Ø LET NN=3Ø: GOSUB 396Ø
167Ø RETURN
168Ø IF INT (RND*18)=1 AND DW=1 THEN
   GOSUB 274Ø: RETURN
169Ø CLS : LET N=1: LET S=1: LET E=1:
   LET W=1: LET F=Ø
17ØØ LET NN=31: GOSUB 396Ø
171Ø IF E$(L,1)<>CHR$ 32 THEN PRINT
   "THERE IS A□";E$(L);"□PASSING.":
   LET F=1
172Ø RETURN
173Ø CLS : LET N=1: LET S=1: LET E=Ø:
   LET W=1: LET U=Ø: LET D=1
174Ø LET NN=32: GOSUB 396Ø
175Ø RETURN
41ØØ DATA 133Ø,138Ø,149Ø,164Ø,168Ø,173Ø,
   155Ø,124Ø,99Ø,1Ø2Ø
```



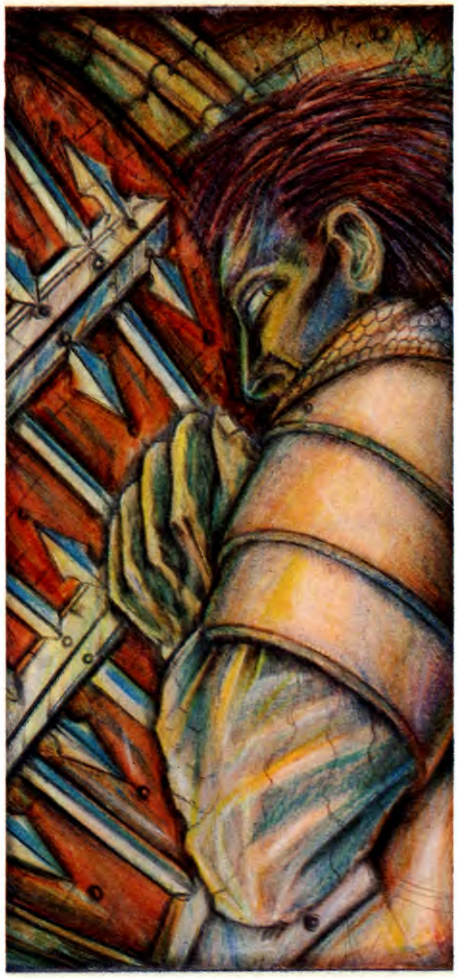
```
411Ø DATA 1Ø8Ø,1Ø6Ø,115Ø,129Ø,144Ø,159Ø,
   116Ø,94Ø,113Ø,118Ø,98Ø,189Ø
```

```
78Ø IF I<1 THEN PRINTLG$"■DON'T
   KNOW HOW TO■"I$:GOTO 61Ø
79Ø IF E$(L)<>""ANDI<>9ANDI<>1Ø
   ANDI<>5ANDI<>12ANDI<>8AND
   F=1 THEN 81Ø
8ØØ GOTO 82Ø
81Ø PRINT LG$"□HE■"E$(L)"■WON'T
   LET YOU.":GOSUB2ØØØØ:GOTO 34Ø
82Ø IF I=1 THEN N$=LEFT$(V$,1)
83Ø IF I=2 THEN GOSUB 219Ø:GOTO 34Ø
84Ø IF I=3 THEN GOSUB 394Ø:GOTO 34Ø
85Ø IF I=4 AND L=3 THEN GOSUB 197Ø
86Ø IF I=5 THEN GOSUB 316Ø:GOTO 34Ø
87Ø IF I=6 THEN GOSUB 4Ø6Ø:GOTO 34Ø
88Ø IF I=7 THEN GOSUB 411Ø:GOTO 34Ø
89Ø IF I=8 THEN GOSUB 425Ø:GOTO 34Ø
9ØØ IF I=9 THEN GOSUB 276Ø:GOTO 34Ø
91Ø IF I=1Ø THEN GOSUB 233Ø:GOTO 34Ø
92Ø IF I=11 THEN GOSUB 421Ø:GOTO 34Ø
93Ø IF I=12 THEN GOSUB 444Ø:GOTO 34Ø
94Ø IF L=1Ø AND N$="W" AND OP=1
   THEN GOSUB 459Ø:GOTO 34Ø
95Ø IFL=1ØANDN$="W"AND PP>ØTHEN
   TX=6Ø:GOSUB99ØØ:GOSUB2ØØØ:GOTO
   34Ø
96Ø IF L=3 AND N$="D" AND D=1 THEN
   GOSUB 197Ø
97Ø IF L=1Ø AND N$="W" THEN
   OP=1:GOSUB 459Ø:GOTO 34Ø
98Ø IF N$="N" AND N=1 THEN
   L=L-3:GOTO 34Ø
99Ø IF N$="S" AND S=1 THEN
   L=L+3:GOTO 34Ø
1ØØØ IF N$="E" AND E=1 THEN
   L=L+1:GOTO 34Ø
1Ø1Ø IF N$="W" AND W=1 THEN
   L=L-1:GOTO 34Ø
1Ø2Ø IF N$="U" AND U=1 THEN
   L=L-6:GOTO 34Ø
1Ø3Ø IF N$="D" AND D=1 THEN
   L=L+6:GOTO 34Ø
1Ø4Ø IF L=16 AND N$="D"AND
   K(7)<>-1 THENTX=29: GOSUB
   99ØØ:GOSUB 2ØØØØ:GOTO34Ø
1Ø5Ø PRINTLG$"□OU CAN'T DO THAT -
   THINK AGAIN!":GOSUB 2ØØØØ:GOTO 34Ø
1Ø6Ø N=Ø:S=Ø:E=Ø:W=1:U=Ø:D=Ø
1Ø7Ø PRINT "■■":TX=2:GOSUB 99ØØ
1Ø79 IF BB<>1 THEN RETURN
1Ø8Ø PRINT:TX=3:GOSUB 99ØØ:PRINT
   LG$TAB(13)"□RESS ANY KEY"
1Ø9Ø PRINT TAB(8)"TO BEGIN YOUR
   ADVENTURE"
11ØØ GET D$:IF D$="" THEN 11ØØ
11Ø5 BB=Ø:RETURN
114Ø PRINT"■":TX=4:GOSUB 99ØØ:
   N=Ø:S=Ø:E=Ø:W=1:U=Ø:D=Ø:
```

```
   RETURN
1150 IF INT(RND(1)*18)+1=6 THEN
   GOSUB 286Ø
116Ø IF INT(RND(1)*18)+1=12 AND
   DW=1 THEN 31ØØ
117Ø PRINT "□":PRINT
118Ø TX=5:GOSUB 99ØØ:N=1:S=Ø:E=Ø:
   W=Ø:U=Ø:D=Ø:RETURN
119Ø IF INT(RND(1)*18)+1=1Ø AND
   DW=1 THEN 31ØØ
12ØØ IF INT(RND(1)*18)+1=1 THEN PRINT
   "□":GOSUB 286Ø
121Ø PRINT "□":PRINT
122Ø TX=6:GOSUB 99ØØ:N=Ø:S=1:E=1:
   W=1:U=1:D=1:IF OP=1 THEN
   TX=62:GOSUB 99ØØ
123Ø RETURN
124Ø IF INT(RND(1)*18)+1=17 AND
   DW=1 THEN 31ØØ
125Ø PRINT "□":PRINT:TX=7:GOSUB
   99ØØ:N=Ø:S=1:E=Ø:W=1:U=1:
   D=Ø:RETURN
126Ø PRINT "□":PRINT:TX=8:GOSUB
   99ØØ:N=Ø:S=1:E=1:W=1:F=Ø:U=Ø:
   D=Ø
127Ø IF E$(L)<>"" THEN TX=9:GOSUB
   99ØØ:GOSUB 2ØØØØ
128Ø IF E$(L)<>"" AND J<8 THEN
   TX=1Ø:GOSUB 99ØØ
129Ø IF E$(L)<>"" AND J>7 THEN
   TX=11:GOSUB 99ØØ:F=1:QQ=QQ+1:
   GOTO 16ØØ
13ØØ RETURN
131Ø IF INT(RND(1)*18)+1=16 THEN
   GOSUB 286Ø
132Ø PRINT "□":TX=28:GOSUB 99ØØ:
   N=1:S=Ø:E=1:W=Ø:U=Ø:D=Ø:
   RETURN
133Ø TX=13:GOSUB 99ØØ:N=1:S=Ø:E=1:
   W=Ø:U=Ø:D=Ø:RETURN
134Ø IF INT(RND(1)*18)+1=14 AND
   DW=1 THEN 31ØØ
135Ø PRINT "□":PRINT:TX=14:GOSUB
   99ØØ
1355 GOSUB2ØØØØ::TX=15:GOSUB99ØØ:
   N=Ø:S=1:E=1:W=Ø:U=Ø:D=Ø:
   RETURN
136Ø IF INT(RND(1)*18)+1=17 THEN
   GOSUB 286Ø
137Ø N=1:S=Ø:E=1:W=1:F=Ø:U=Ø:
   D=Ø
138Ø IF G<=4 THEN E$(L)=""
139Ø PRINT"□":PRINT:TX=16:GOSUB
   99ØØ
14ØØ IF E$(L)<>"" THEN PRINT:PRINT
   "□E STIRS.":F=1
141Ø RETURN
142Ø IF INT(RND(1)*18)+1=5 THEN
   GOSUB 286Ø
143Ø PRINT "□"
144Ø N=1:S=Ø:E=Ø:W=1:U=Ø:D=Ø
145Ø PRINT:TX=17:GOSUB 99ØØ
146Ø RETURN
147Ø N=1:S=Ø:W=1:E=1:F=Ø:U=Ø:
   D=Ø
148Ø PRINT:TX=18:GOSUB 99ØØ
149Ø IF K(19)=-1 THEN F=Ø:TX=19:
   GOSUB 99ØØ
15ØØ RETURN
151Ø IF INT(RND(1)*18)+1=1 AND
   DW=1 THEN 31ØØ
152Ø PRINT "□"
153Ø N=Ø:S=1:E=1:W=Ø:U=Ø:D=Ø
154Ø PRINT:TX=2Ø:GOSUB 99ØØ
155Ø RETURN
156Ø PRINT "□"
157Ø N=Ø:S=1:E=1:W=1:U=Ø:D=Ø
158Ø PRINT:TX=21:GOSUB 99ØØ
159Ø IF E$(L)<>"" THEN TX=22:GOSUB
   99ØØ:F=1:QQ=QQ+1
16ØØ IF QQ>4 THEN PRINT "□HE
   SCREAMS!":GOSUB 2ØØØØ
161Ø IF QQ>4 THEN PRINT"□WO
   GUARDS APPEAR.":GOSUB 2ØØØØ
162Ø IF QQ>4 THEN PRINT"□OU
   SURRENDER.":GOSUB 2ØØØØ:GOTO
   1ØØØØ
163Ø RETURN
164Ø N=1:S=Ø:E=Ø:W=1:F=Ø:U=Ø:
   D=Ø
165Ø TX=23:GOSUB 99ØØ
166Ø TX=24:GOSUB 99ØØ:F=1
```

```
54Ø IF I$=t$ AND t=Ø OR I$=i$ AND
   i=Ø THEN PROCI:GOTO 27Ø
55Ø I=INSTR(I$,"□")
57Ø V$=LEFT$(I$,I-1)
58Ø T$=MID$(I$,I+1)
59Ø IF V$="go" THEN V$=T$
6ØØ asc$="":FORc=1TOLEN(T$):asc=ASC
   (MID$(T$,c,1))
61Ø IFasc<91 ANDasc<>32 THEN
   asc=asc+32
62Ø asc$=asc$+CHR$(asc)
63Ø NEXT:T$=asc$
64Ø I=Ø
65Ø FOR H=1 TO 32
66Ø IFINSTR(R$(H),V$)=1THENI=R(H)
67Ø NEXT
68Ø IF I<1 THEN PRINT"I don't know how
   to□"I$:GOTO 53Ø
69Ø IF E$(L)<>"" AND I<>9 AND
   I<>1Ø AND I<>5 AND I<>12 AND
   I<>8 AND F=1 THEN
   PRINT"The□"E$(L)"□won't let you":
   D=INKEY(25Ø):GOTO 27Ø
7ØØ IF I=1 THEN N$=LEFT$(V$,1)
71Ø IF I=2 THEN PROCC:GOTO 27Ø
72Ø IF I=3 THEN PROCJ:GOTO 27Ø
73Ø IF I=4 AND L=3 THEN PROCA
74Ø IF I=5 THEN PROCH:GOTO 27Ø
75Ø IF I=6 THEN PROCK:GOTO 27Ø
76Ø IF I=7 THEN PROCL:GOTO 27Ø
77Ø IF I=8 THEN PROCN:GOTO27Ø
78Ø IF I=9 THEN PROCE:GOTO 27Ø
79Ø IF I=1Ø THEN PROCD:GOTO 27Ø
8ØØ IF I=11 THEN PROCM:GOTO 27Ø
81Ø IF I=12 THEN PROCO:GOTO 27Ø
82Ø IFL=1ØANDN$="w"ANDOP=1THEN
   PROCP:GOTO27Ø
83Ø IFL=1ØANDN$="w"AND p>ØTHEN
   PRINTFNW(6Ø):D=INKEY(4ØØ):GOTO27Ø
84Ø IFL=3ANDN$="d"ANDD=1PROCA
85Ø IFL=1ØANDN$="w"THENOP=1:
   PROCP:GOTO27Ø
86Ø IF N$="n" AND N=1 THEN
   L=L-3:GOTO 27Ø
87Ø IF N$="s" AND S=1 THEN
   L=L+3:GOTO 27Ø
88Ø IF N$="e" AND E=1 THEN
   L=L+1:GOTO 27Ø
89Ø IF N$="w" AND W=1 THEN
   L=L-1:GOTO 27Ø
9ØØ IF N$="u" AND U=1 THEN
   L=L-6:GOTO 27Ø
91Ø IF N$="d" AND D=1 THEN
   L=L+6:GOTO 27Ø
92Ø IFL=16ANDN$="d"ANDK(7)<>-1
   THENPRINTFNW(29)
93Ø PRINT"You can't do that here think
   again!":VDU7:D=INKEY(25Ø):GOTO 27Ø
94Ø N=Ø:S=Ø:E=Ø:W=1:U=Ø:D=Ø
95Ø CLS:PRINT''FNW(2)
96Ø IF b=1 THEN PRINT'FNW(3)'CHR$(129)
   "PRESS ANY KEY TO BEGIN YOUR
   ADVENTURE":D$=GET$:b=Ø
97Ø RETURN
98Ø CLS:PRINT''FNW(4):N=Ø:S=Ø:E=Ø:
   W=1:U=Ø:D=Ø:RETURN
99Ø IF RND(18)=6 THEN PROCF
1ØØØ IF RND(18)=12 AND dw=1 THEN
   PROCG:RETURN
1Ø1Ø CLS:PRINT'FNW(5):N=1:S=Ø:E=Ø:
   W=Ø:U=Ø:D=Ø:RETURN
1Ø2Ø IF RND(18)=1Ø AND dw=1 THEN
   PROCG:RETURN
1Ø3Ø IF RND(18)=1 THEN CLS:PROCF
1Ø4Ø CLS:PRINT''FNW(6):N=Ø:S=1:E=1:
   W=1:U=1:D=1:IFOP=1THEN PRINT
   FNW(62)
1Ø5Ø RETURN
1Ø6Ø IFRND(18)=17 AND dw=1 THEN
   PROCG:RETURN
1Ø7Ø CLS:PRINT''FNW(7):N=Ø:S=1:E=Ø:
   W=1:U=1:D=Ø:RETURN
1Ø8Ø CLS:PRINT''FNW(8):N=Ø:S=1:E=1:
   W=1:F=Ø:U=Ø:D=Ø
1Ø9Ø IF E$(L)<>""THEN PRINTFNW(9):
   D=INKEY(1ØØ)
11ØØ IF E$(L)<>"" AND J<8 THEN
   PRINTFNW(1Ø)
111Ø IF E$(L)<>"" AND J>7 THEN
```

```
    PRINT FNW(11): F=1: qq=qq+1:
    GOTO142Ø
112Ø RETURN
113Ø IF RND(18)=16 THEN PROCF
114Ø CLS:PRINT'FNW(28):N=1:S=Ø:E=1:
    W=Ø:U=Ø:D=Ø:RETURN
115Ø PRINT'FNW(13):N=1:S=Ø:E=1:
    W=Ø:U=Ø:D=Ø:RETURN
116Ø IFRND(18)=14 AND dw=1 THEN
    PROCG:RETURN
117Ø CLS:PRINT''FNW(14):D=INKEY(3ØØ):
    PRINT'FNW(15):N=Ø:S=1:E=1:W=Ø:
    U=Ø:D=Ø:RETURN
118Ø IF RND(18)=17 THEN PROCF
119Ø N=1:S=Ø:E=1:W=1:F=Ø:
    U=Ø:D=Ø
12ØØ IF G<=4 THEN E$(L)=""
121Ø CLS:PRINT''FNW(16)
122Ø IF E$(L)<>""THEN PRINT'"He
    stirs":F=1
123Ø RETURN
124Ø IF RND(18)=5 THEN PROCF
125Ø CLS
126Ø N=1:S=Ø:E=Ø:W=1: U=Ø:D=Ø
127Ø PRINT''FNW(17)
128Ø RETURN
129Ø N=1:S=Ø:W=1:E=1:F=Ø:
    U=Ø:D=Ø
13ØØ PRINT''FNW(18)
131Ø IFK(19)=-1THEN  F=Ø: PRINT
    FNW(19)
132Ø RETURN
133Ø IF RND(18)=1 AND dw=1 THEN
    PROCG:RETURN
134Ø CLS
135Ø N=Ø:S=1:E=1:W=Ø:U=Ø:
    D=Ø
136Ø PRINT'FNW(2Ø)
137Ø RETURN
138Ø CLS
139Ø N=Ø:S=1:E=1:W=1:U=Ø:
    D=Ø
14ØØ PRINT''FNW(21)
141Ø IF E$(L)<>"" THEN PRINTFNW(22):
    F=1:qq=qq+1
142Ø IFqq>4THENPRINT"She screams.":D=
    INKEY(1ØØ):PRINT"Two guards appear.":
    D=INKEY(1ØØ):PRINT"You surrender.":
    D=INKEY(1ØØ):END
143Ø RETURN
144Ø N=1:S=Ø:E=Ø:W=1:F=Ø:
    U=Ø:D=Ø
145Ø PRINTFNW(23)
146Ø PRINTFNW(24):F=1
147Ø IFRND(18)<4THENF=Ø:
    PRINTFNW(63)
148Ø RETURN
149Ø IF RND(18)=3 AND dw=1 THEN
    PROCG:RETURN
15ØØ CLS
151Ø N=Ø:S=1:E=Ø:W=1:U=Ø:D=Ø
```



```
152Ø PRINT''FNW(25)
153Ø IF K(17)=-1 THEN PRINTFNW(26):
    D=1
154Ø RETURN
155Ø CLS
156Ø N=1:S=Ø:E=1:W=Ø:U=Ø:D=Ø
157Ø PRINT''FNW(27)
158Ø RETURN
159Ø CLS
16ØØ N=Ø:S=1:E=Ø:W=Ø:U=1:D=Ø
161Ø PRINT''FNW(12)
162Ø IFK(7)=-1THENPRINTFNW(64):D=1
163Ø RETURN
164Ø IF RND(18)=1 THEN PROCF
165Ø CLS:N=1:S=1:E=1:W=Ø:U=Ø:D=1
```

Tandy owners should take care to change Line 96Ø. EXEC41194 should be altered to EXEC36Ø38.

```
51Ø IF K(C7)=L AND C7<>L AND C7<>7
    THEN PRINT"THE□";O$(C7);"□IS HERE"
52Ø NEXT
53Ø INPUT "WHAT NOW";I$
54Ø IF I$=T7$ AND T7=Ø OR I$=I7$ AND
    I7=Ø THEN GOSUB 3Ø4Ø:GOTO27Ø
55Ø I=INSTR(I$,"□")
56Ø IF I=Ø THEN V$=I$:GOTO 58Ø
57Ø IF(I-1)<1THENV$=""ELSE V$=
    LEFT$(I$,I-1)
58Ø T$=MID$(I$,I+1)
59Ø IF V$="GO" THEN V$=T$
6ØØ REM
64Ø I=Ø
65Ø FOR H=1 TO 32
66Ø IF INSTR(R$(H),V$)=1 THEN I=R(H)
67Ø NEXT
68Ø IF I<1 THEN PRINT"I DON'T KNOW
    HOW TO□";I$:GOTO53Ø
69Ø IF E$(L)<>"" AND I<>9 AND
    I<>1Ø AND I<>5 AND I<>12 AND
    I<>8 AND F=1 THEN
    PRINT"THE□";E$(L); "□WON'T LET
    YOU":GOSUB55ØØ:GOTO27Ø
7ØØ IF I=1 THEN N$=LEFT$(V$,1)
71Ø IF I=2 THEN GOSUB 197Ø:GOTO27Ø
72Ø IF I=3 THEN GOSUB 33ØØ:GOTO27Ø
73Ø IF I=4 AND L=3 THEN GOSUB 176Ø
74Ø IF I=5 THEN GOSUB 28ØØ:GOTO27Ø
75Ø IF I=6 THEN GOSUB 339Ø:GOTO27Ø
76Ø IF I=7 THEN GOSUB 344Ø:GOTO27Ø
77Ø IF I=8 THEN GOSUB 356Ø:GOTO27Ø
78Ø IF I=9 THEN GOSUB 245Ø:GOTO27Ø
79Ø IF I=1Ø THEN GOSUB 2Ø9Ø:GOTO27Ø
8ØØ IF I=11 THEN GOSUB 352Ø:GOTO27Ø
81Ø IF I=12 THEN GOSUB 373Ø:GOTO27Ø
82Ø IF L=1Ø AND N$="W" AND OP=1
    THEN GOSUB 381Ø:GOTO27Ø
83Ø IF L=1Ø AND N$="W" AND P7>Ø
    THEN WN=6Ø:GOSUB51ØØ:
    GOSUB55ØØ:GOTO27Ø
84Ø IF L=3 AND N$="D" AND D=1 THEN
    GOSUB 176Ø
85Ø IF L=1Ø AND N$="W" THEN
    OP=1:GOSUB 381Ø:GOTO 27Ø
86Ø IF N$="N" AND N=1 THEN
    L=L-3:GOTO 27Ø
87Ø IF N$="S" AND S=1 THEN
    L=L+3:GOTO27Ø
88Ø IF N$="E" AND E=1 THEN
    L=L+1:GOTO27Ø
89Ø IF N$="W" AND W=1 THEN
    L=L-1:GOTO27Ø
9ØØ IF N$="U" AND U=1 THEN
    L=L-6:GOTO27Ø
91Ø IF N$="D" AND D=1 THEN
    L=L+6:GOTO27Ø
92Ø IF L=16 AND N$="D" AND
    K(7)<>-1 THEN WN=29:GOSUB
    51ØØ:GOSUB55ØØ:GOTO27Ø
93Ø PRINT"YOU CAN'T DO THAT HERE!":
    SOUND1,1:GOSUB55ØØ:GOTO27Ø
94Ø N=Ø:S=Ø:E=Ø:W=1:U=Ø:D=Ø
95Ø CLS:WN=2:GOSUB51ØØ
96Ø IF B7=1 THEN WN=3:GOSUB51ØØ:
    PRINT"PRESS ANY KEY TO BEGIN YOUR
```

```
    □□□□□ADVENTURE":EXEC41194:
    B7=Ø
970 RETURN
980 CLS:WN=4:GOSUB5100:N=Ø:S=Ø:
    E=Ø:W=1:U=Ø:D=Ø:RETURN
990 IF RND(18)=6 THEN GOSUB 255Ø
1000 IF RND(18)=12 AND DW=1 THEN
    GOSUB 274Ø:RETURN
1010 CLS:WN=5:GOSUB51ØØ:N=1:S=Ø:
    E=Ø:W=Ø:U=Ø:D=Ø:RETURN
1020 IF RND(18)=1Ø AND DW=1 THEN
    GOSUB 274Ø:RETURN
1030 IF RND(18)=1 THEN CLS:GOSUB 255Ø
1040 CLS:WN=6:GOSUB51ØØ:N=Ø:S=1:
    E=1:W=1:U=1:D=1:IF OP=1 THEN
    WN=62:GOSUB51ØØ
1050 RETURN
1060 IF RND(18)=17 AND DW=1 THEN
    GOSUB 274Ø:RETURN
1070 CLS:WN=7:GOSUB51ØØ:N=Ø:S=1:
    E=Ø:W=1:U=1:D=Ø:RETURN
1080 CLS:WN=8:GOSUB51ØØ:N=Ø:S=1:
    E=1:W=1:F=Ø:U=Ø:D=Ø
1090 IF E$(L)<>"" THEN WN=9:GOSUB
    51ØØ:GOSUB55ØØ
1100 IF E$(L)<>"" AND J<8 THEN
    WN=1Ø:GOSUB51ØØ
1110 IF E$(L)<>"" AND J>7 THEN
    WN=11:GOSUB51ØØ:F=1:
    QQ=QQ+1:GOTO142Ø
1120 RETURN
1130 IF RND(18)=16 THEN GOSUB 255Ø
1140 CLS:WN=28:GOSUB51ØØ:N=1:S=Ø:
    E=1:W=Ø:U=Ø:D=Ø:RETURN
1150 WN=13:GOSUB51ØØ:N=1:S=Ø:
    E=1:W=Ø:U=Ø:D=Ø:RETURN
1160 IF RND(18)=14 AND DW=1 THEN
    GOSUB 274Ø:RETURN
1170 CLS:WN=14:GOSUB51ØØ:GOSUB55ØØ:
    WN=15:GOSUB51ØØ:N=Ø:S=1:E=1:
    W=Ø:U=Ø:D=Ø:RETURN
1180 IF RND(18)=17 THEN GOSUB 255Ø
1190 N=1:S=Ø:E=1:W=1:F=Ø:
    U=Ø:D=Ø
1200 IF G<=4 THEN E$(L)=""
1210 CLS:WN=16:GOSUB51ØØ
1220 IF E$(L)<>"" THEN PRINT"HE
    STIRS":F=1
1230 RETURN
1240 IF RND(18)=5 THEN GOSUB 255Ø
1250 CLS
1260 N=1:S=Ø:E=Ø:W=1:U=Ø:D=Ø
1270 WN=17:GOSUB 51ØØ
1280 RETURN
1290 N=1:S=Ø:W=1:E=1:F=Ø:
    U=Ø:D=Ø
1300 WN=18:GOSUB51ØØ
1310 IF K(19)=-1 THEN F=Ø:WN=19:
    GOSUB 51ØØ
1320 RETURN
1330 IF RND(18)=1 AND DW=1 THEN
    GOSUB 274Ø:RETURN
1340 CLS
1350 N=Ø:S=1:E=1:W=Ø:U=Ø:D=Ø
1360 WN=2Ø:GOSUB51ØØ
1370 RETURN
1380 CLS
1390 N=Ø:S=1:E=1:W=1:U=Ø:D=Ø
1400 WN=21:GOSUB 51ØØ
1410 IF E$(L)<>"" THEN WN=22:GOSUB
    51ØØ:F=1:QQ=QQ+1
1420 IF QQ>4 THEN PRINT"SHE SCREAMS.":
    GOSUB 55ØØ:PRINT"TWO GUARDS
    APPEAR.":GOSUB55ØØ:PRINT "YOU
    SURRENDER.":GOSUB 55ØØ:GOTO 65ØØ
1430 RETURN
1440 N=1:S=Ø:E=Ø:W=1:F=Ø:
    U=Ø:D=Ø
1450 WN=23:GOSUB 51ØØ
1460 WN=24:GOSUB51ØØ:F=1
1470 IF RND(18)<4 THEN
    F=Ø:WN=63:GOSUB51ØØ
1480 RETURN
1490 IF RND(18)=3 AND DW=1 THEN
    GOSUB 274Ø:RETURN
1500 CLS
1510 N=Ø:S=1:E=Ø:W=1:U=Ø:D=Ø
1520 WN=25:GOSUB51ØØ
1530 IF K(17)=-1 THEN WN=26:
    GOSUB51ØØ:D=1
1540 RETURN
1550 CLS
1560 N=1:S=Ø:E=1:W=Ø:U=Ø:D=Ø
1570 WN=27:GOSUB51ØØ
1580 RETURN
1590 CLS
1600 N=Ø:S=1:E=Ø:W=Ø:U=1:D=Ø
1610 WN=12:GOSUB51ØØ
1620 IF K(7)=-1 THEN WN=64:
    GOSUB51ØØ:D=1
1630 RETURN
1640 IF RND(18)=1 THEN GOSUB 255Ø
1650 CLS:N=1:S=1:E=1:W=Ø:U=Ø:D=1
1660 WN=3Ø:GOSUB51ØØ
1670 RETURN
1680 IF RND(18)=1 AND DW=1 THEN
    GOSUB 274Ø:RETURN
1690 CLS:N=1:S=1:E=1:W=1:F=Ø
1700 WN=31:GOSUB51ØØ
1710 IF E$(L)<>"" THEN PRINT"THERE
    IS A ";E$(L);" PASSING":F=1
1720 RETURN
1730 CLS:N=1:S=1:E=Ø:W=1:U=Ø:D=1
1740 WN=32:GOSUB51ØØ
1750 RETURN
1760 REM *** Proc a
1770 CLS
```

CONSTRUCTING A LISP PROGRAM

Constructing and manipulating lists is the heart of LISP. But before you can put together a program, find out how to structure the process around the available functions

As you will have discovered from the first article on LISP (pages 1410 to 1415), everything in this language is done by functions. And for the programmer who is used to working in BASIC, getting used to this is the hardest part of becoming fluent in LISP.

You have already seen examples of quite a number of LISP's standard functions. But in order to write proper LISP programs, you need to define your own functions in addition to those already built into the LISP system. This is similar to the way in which you saw LOGO used in the first articles in this series—where LOGO's primitives were used as the basis of new procedures, which could then be built up into even more complex programs. But the process is rather different.

DEFINING A FUNCTION

Just like everything else in LISP, defining a function requires the use of a function—this is called DEFUN. If, for instance, you want to define a function called PLUS1 which adds 1 to whatever you feed it, in LISP, this means that your function has to take a single atom in its definition and add one to its value. Using DEFUN this could be done by giving the following s-expression to LISP:

```
(DEFUN PLUS1 (A) (PLUS 1 A)).
```

DEFUN has three *arguments* (lists on which it has to work), which it does not evaluate. The first argument is the name of the function, in this case PLUS1. The next argument is a LISP list of the function's arguments—in this example, there is only one, the atom A. The last argument of DEFUN is a LISP s-expression to be evaluated. The result of the function is the value of this s-expression.

When it evaluates the s-expression, LISP substitutes in it the values given to the function's arguments. So (PLUS1 2) gives 3 because LISP substitutes 2 for the atom A and then evaluates the s-expression. In most implementations of LISP, any number of LISP s-expressions can follow the argument list and the value of the function is given by the value of the last one.

These functions are quite similar to those used in BASIC. For instance, in BASIC you could define a function to do the above job by:

```
DEFFNPLUS 1(A)=A+1
```

However, as usual, LISP functions are more powerful—a LISP function can take lists as its arguments and can return a list as its value. The arguments of this type of user-defined function are evaluated before being substituted in the function's definition, so (PLUS1 (PLUS1 1)) evaluates to 3. The inner call of PLUS1 gives 2 and this is then fed to the outer PLUS1, giving the value of 3. It is also possible to define LISP functions that do not evaluate their arguments and functions that can have an arbitary number of arguments.

Usually, the value of DEFUN is the *name* of the function defined. Thus the interaction with LISP when you define a function looks like this:

```
Evaluate: (DEFUN PLUS1 (A) (PLUS 1 A))
Value is: PLUS1
```

Suppose you now type the name of the function in response to the Evaluate prompt:

```
Evaluate: PLUS1
Value is: (LAMBDA (A) (PLUS 1 A))
```

As you can see, LISP replies with the definition of the function, which is now the value of the atom that is the function's name. The word LAMBDA is a special atom used by LISP to denote the fact that the s-expression is a function definition. Its effect above is to cause a substitution for the value of A in the following s-expression. It is important to understand the difference between evaluating the name of the function like this and actually using the function. When you want to call the function it is surrounded by a pair of brackets:

```
Evaluate: (PLUS1 Ø)
Value is: 1
```

It is possible to find the definitions of some of the functions built into the LISP system in a similar way, although, because most of them are written in machine code, their definitions cannot be displayed.

Another example of a user-defined function is (DEFUN SECOND (LI) (CAR (CDR LI))). This function takes a list as its argument and returns the second element in it. Thus the value of (SECOND '(tea sugar milk)) is sugar. As

- STRUCTURING A PROGRAM
- TOP-DOWN DESIGN
- EQUALITY TESTING
- TAIL RECURSION
- COMPLEX FUNCTIONS
- DATA STRUCTURES
- WORKING MEMORY
- THE FREE LIST
- GARBAGE COLLECTION
- DEFINING A FUNCTION

you may remember, exactly the same effect could be achieved by using the built-in function CADR. In this example, the atom LI is used as the name of the arguments, but you can use any name you like for the arguments.

STRUCTURING A PROGRAM

From the examples so far, you might decide that a LISP computer is no more than a sophisticated desk calculator, simply replying with the values of expressions. This is quite different to the behaviour expected from a computer where you want a program to go on continuously until you tell it to stop.

To write a LISP program a top-down style of programming is used. You have already seen this principle applied in Pascal programs, which have to be written this way, and the process is similar in LISP. The task that the program is to do is split up into further simpler tasks which are in turn split up. This process is repeated until each subtask is so simple that it can be done by a single LISP function. Each of these subtasks is called by another function representing the task which was split up to obtain them. Eventually, the entire program is represented by a single function which calls all the others. To set the program off (the equivalent of RUN in BASIC) you get LISP to evaluate this master function.

An advantage of this style of programming is that it is very easy to test each of the functions as they are written, by typing them in directly from the keyboard along with any necessary data. Doing this makes finding errors in programs much easier.

REPETITION

The final ingredient which you need to write complete programs is a method of repeating tasks a number of times. Technically speaking, there are two ways of doing this—iteration and recursion. Iteration is very familiar to users of BASIC, the classic example being the FOR...NEXT loop. And if you have read the article on pages 1289 to 1295, you will have seen how recursion can be applied in BASIC programs. To demonstrate the techniques used in LISP, an example of a program to calculate factorials is going to be used. The factorial of an integer is calculated by multiplying together all the integers between that number and one, so 5 factorial is 5 × 4 × 3 × 2 × 1 = 120. If you wanted to calculate the factorials of some numbers in BASIC, you would probably define the following function:

```
DEFFNFACT(N)
F=1
FOR I=1 TO N
F=F*I
NEXT
=F
```

As you can see, FNFACT(5) equals 120 as it should do. The point here is that the factorial is calculated by iteration—the function keeps going through the loop, modifying F until it reaches the desired answer. Most languages, including BASIC, rely on iterative constructs like the FOR...NEXT and REPEAT...UNTIL loops when it is necessary to do a task several times. However, LISP is different. Instead of iteration, LISP's main way of repeating things is recursion.

RECURSION

You are probably already familiar with recursive procedures in BASIC. But as recursion is so important in LISP, it is worth a quick recap on the fundamental principles.

Recursion means that the solution of a problem is defined in terms of itself. This may seem a bit paradoxical, because if the solution is defined in terms of itself, then it seems as if a solution will never be reached, and instead there will just be an infinite spiral of solutions each of which refers to another. Actually, the problem is that this definition is not rigorous enough. A proper recursive definition consists of two parts:

(a) An exceptional case for which the solution is known.

(b) A definition of the solution of the problem in terms of the solution of a simpler version of the problem.

Here (a) is the crucial part, for without this, a recursive definition would indeed go on for ever and be of no use.

Using this definition and going back to the example of the factorial program, you could write a recursive definition of N factorial as:

(a) 1 factorial is 1.

(b) N factorial is given by N times N−1 factorial.

This definition can now be turned into a LISP function which can be written as:

```
(DEFUN FACT (N) (COND ((EQ N 1) 1)
(T (TIMES (FACT (DIFFERENCE N 1)) N))))
```

After typing in this definition (FACT 5) will be evaluated to 120 by LISP. This definition uses the COND function described in the first article on LISP. The first s-expression of the first clause checks to see if N is one. If it is, the value of the COND is given by the second s-expression; in this case 1. Otherwise LISP carries on to the second clause and since the first s-expression is T, it finds that the value of the COND is the expression:

```
(TIMES (FACT (DIFFERENCE N 1)) N)
```

which in English is N times N−1 factorial.

When LISP calculates 5 factorial, the function FACT calls itself 4 times with a different value for the argument N each time. The way in which LISP substitutes the value of the function's argument into the function's definition is cleverly arranged so that there is no confusion between these different calls of the FACT function, despite the fact that there is more than one version of it active at a time.

Recursive definitions require a different way of thinking to that which you may usually use when writing BASIC programs. However, recursion is a very useful technique and it allows programs to be written simply that would be very complex to write using iterative ideas. Although BASIC is not really very suited to recursive programming, this is why even some versions of BASIC now use it. For instance, in BBC BASIC you could define the factorial function using recursion:

```
DEFFNFACT(N)
IF N=1 THEN=1 ELSE=N *FNFACT(N-1)
```

Obviously, this is very similar to the LISP.

Although recursion is a very useful concept, it is not always convenient, and many programmers prefer the iterative way of calculating factorials to the recursive method. In 'pure' LISP, recursion is the only available technique. However, over the years since LISP was invented, people have tried to graft onto it the iterative ideas of other languages. As time has gone by, some of these have been dropped. This makes it difficult to describe the iterative constructs any particular version of LISP may have. For instance, some modern versions have REPEAT...UNTIL and REPEAT...WHILE. Older ones may have forms of FOR...NEXT or even GOTO. However the kind of sophisticated list processing problems for which LISP was designed are usually best solved by recursive techniques.

EQUALITY TESTING

An example of a more complex LISP function definition is provided by the function EQUAL. You have already seen the function EQ that tests for the equality of two atoms. The function EQUAL tests to see if two s-expressions are the same. If they are, it returns the value T—otherwise it has the value NIL. Some LISP implementations have this function built in—if yours does not you must define it yourself. Obviously, defining this function is a difficult problem because it has to take into account the complex structures that the two lists to be compared might have. In English, you can define the function as:

(a1) If EQ applied to the arguments has the value T then EQUAL has the value T.

(a2) If either argument is an atom and (a1) was not satisfied when EQUAL has the value NIL.

(b) If the value of EQUAL applied to the CARs of the original arguments is T and if the value of EQUAL applied to CDRs of the original arguments is T then EQUAL has the value T otherwise it has the value NIL.

In Lisp the definition of EQUAL is:

```
(DEFUN EQUAL (L1 L2) (COND ((EQ L1
   L2) T)
((OR (ATOM L1) (ATOM L2)) NIL)
((EQUAL (CAR L1)(CAR L2))(EQUAL (CDR
   L1)(CDR L2)))))
```

The function is given two s-expressions L1 and L2 to compare. If L1 and L2 are the same atom then (EQ L1 L2) will be T and the COND function will terminate with the value T. Now if either L1 or L2 is an atom and the EQ did not have the value T then L1 and L2 cannot be the same and EQUAL must have the value NIL. In the second clause of the COND the ATOM and OR functions are used implement this condition.

If L1 and L2 pass through the first two tests then they must be lists. In order to compare them, they must be split up and then EQUAL can be used on their constituent parts. The natural way of doing this is, of course, with the functions CAR and CDR. Here a trick is used in the last line of the COND. First the two CARs are compared and if they are the same, EQUAL will return T and LISP will move onto the second s-expression where EQUAL is applied to the two CDRs. The value of the function is then the value of this EQUAL.

This is correct if the first EQUAL had the value T. If the first EQUAL has the value NIL then LISP tries to move on to the next clause, and finding there isn't one it automatically gives the COND the value NIL.

A rather clearer but less concise way of doing things would be to make the last two clauses of the COND into:

```
((AND (EQUAL (CAR L1)(CAR L2))(EQUAL
   (CDR L1)(CDR L2))) T) (T NIL)
```

You may wonder why the last line of the COND was not written as:

```
(T (EQUAL L1 L2))
```

This would be a very bad recursive definition because it goes on forever. The point about the original definition is that as the function recurses, the use of the CAR and CDR functions makes the lists which are the arguments of EQUAL get simpler. Eventually, they become atoms and then the first two 'exceptional cases' in the COND terminate the recursion. Using this new definition L1 and L2 never become any simpler and so the function goes on calling itself for ever.

TAIL RECURSION

In the factorial function, after the function has called itself and got an answer back, it must multiply this by N before passing the result back to the level above. In a *tail recursive* definition, a function passes back the results of any deeper levels of recursion without change. This uses the computer more efficiently. The following is a tail recursive definition of the factorial function:

```
(DEFUN FACT (N) (FACTX N 1))
(DEFUN FACTX (N F) (COND ((EQ N 1) F)
(T (FACTX (DIFFERENCE N 1)(TIMES F N)))))
```

This requires two functions and is quite similar in spirit to the original iterative definition of the factorial function:

COMPLEX FUNCTIONS

In addition to the basic functions like CAR and CDR built-in to the LISP system, there are some rather more advanced ones. A good example is the MAPCAR function. This has two arguments—the first is a function and the second a list. MAPCAR applies the function to each element in the list and returns a list of the results. So for instance, the result of evaluating (MAPCAR 'PLUS1'(1 2 3 4 5)) is (2 3 4 5 6).

The last function to look at is perhaps the most important function in LISP. It is called EVAL and when you type something in at the keyboard of a LISP system it is evaluated using this function. However, it is possible for a LISP function to generate a valid LISP s-expression and then evaluate it using EVAL. For instance, (PLUS 1 1) evaluates to 2 and the expression '(PLUS 1 1) will evaluate to (PLUS 1 1). But (EVAL '(PLUS 1 1)) evaluates to 2. So in a sense EVAL is the opposite of the single quote. EVAL then contains all the power of LISP, but surprisingly enough it is quite easy to write down its definition in LISP.

DATA STRUCTURES

Although defining functions is essential to writing programs, in LISP data structures are almost as important—as in any language.

In BASIC, there are really only two data structures—the string and the array. When you write a BASIC program, you must decide how to represent the data it will manipulate in terms of these two structures, and choosing the best way to represent the data in a problem can often make the solution of the

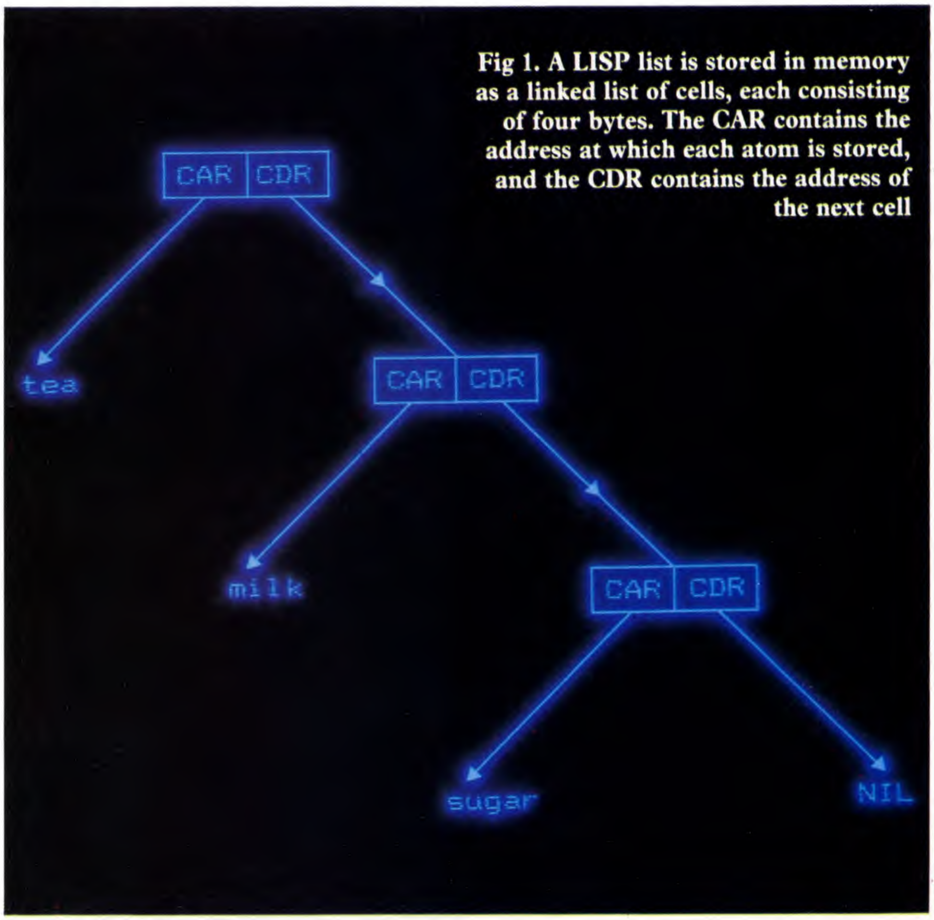


Fig 1. A LISP list is stored in memory as a linked list of cells, each consisting of four bytes. The CAR contains the address at which each atom is stored, and the CDR contains the address of the next cell

problem itself much easier.

In LISP, the concept of lists allows the construction of an unlimited number of types of data structure—it is up to the LISP programmer to use this freedom to maximum advantage. To give a simple example of how rapidly LISP data structures become more complex than in BASIC, a list of atoms may be considered as similar to an array. However, it is possible to have a list each of whose elements is a list of atoms—in other words an array of arrays.

An important LISP data structure is the property list. As the previous article showed, a LISP atom can have a value. However, in addition to its value, an atom can also have 'properties'. For instance, suppose you have the atom FRED and that this represents a person's name. This atom can have properties corresponding to all the information you know about the person. For example (PUT 'FRED 'AGE 32), will give the value 32 to the property AGE of the atom FRED. Next you might want to add details of the person's height: this could be done with (PUT 'FRED 'HEIGHT 200). The values of the properties can be returned with the GET function, so (GET 'FRED 'AGE) has the value 32.

The uses of properties are only limited by the imagination of the LISP user. A possible use is as tags. Suppose you have two lists—one called LEFT contains the colours of all your left socks (red green blue yellow) the other, called RIGHT, contains the colours of all your right socks (pink orange black green). The problem is to find if you have any matching pairs. One way of tackling it would be to take one sock from LEFT and compare it with every sock in the list RIGHT and then repeat this for all the other socks in LEFT. For very large lists this would be a very slow process. Instead, you can use property lists. First go through RIGHT attaching the property SEEn with the value NIL to each sock e.g. (PUT 'pink 'SEEN NIL). Now go through the list LEFT and put the property SEEN with the value T on each sock. Last, again go through the list RIGHT and look at the value of the property SEEN on each sock e.g. (GET 'pink 'SEEN); those which have the value of T must have a matching sock in the list LEFT. The search this way is much faster.

WORKING MEMORY

You can make even more use of properties thanks to functions which allow LISP to manipulate them independently. REMPROP REMoves a PROPerty from an atom, as in (REMPROP 'FRED 'AGE). In addition, there is often a function PLIST which has as its value the Property LIST for an atom. For instance (PLIST 'FRED) would have the value ((AGE . 32) (HEIGHT . 200)). Each item in this list is a dotted pair (briefly referred to in the previous article). To understand this idea, you must understand what is going on behind the scenes in a LISP computer.

Obviously, when a LISP system is working, lists are being manipulated all the time. But most computers have no special instruction for doing this. This means that the lists must be represented in terms of the usual contents of a computer. For instance, the typical home computer has about 64,000 memory locations each of which can hold a single 8 bit number. Actually holding the address of a memory location requires a 16 bit number.

In a typical LISP system, all the memory locations in the computer (to be used for storing lists) would be split up into cells. Each cell consists of four bytes—two 16 bit words which can each hold the address of another cell. The first two bytes of a cell can be called its CAR and the second two bytes, its CDR. A LISP list is stored as a linked list of such cells. Consider the list (tea milk sugar). Each item in the list is represented by a cell. The CAR of the cell contains the address at which the appropriate atom is stored and the CDR contains the address of the next cell in the list. Fig 1. should make this clear.

The question arises what happens at the end of the list? Where does the last CDR of the linked list point to? Usually, it points to the special atom NIL as shown in the diagram. A dotted pair is a representation of one cell in the linked list. This means that a list with one item (FRED), say, can be written in LISP as (FRED . NIL). So dotted pairs are LISP's way of displaying how lists are stored. Suppose you typed (CONS 'FRED 'SMITH). LISP would reply with (FRED . SMITH). In memory you would have created the structure in Fig 2.

Fig 2. A structure stored in memory like this is displayed by LISP as a dotted pair

The CAR, CDR and CONS operators can also be applied to dotted pairs. For example, (CAR '(FRED . SMITH)) is FRED. These dotted pairs can be useful, but it is possible to write LISP programs without worrying about them.

THE FREE LIST

When you begin a programming session on a LISP computer, all the free memory is divided up into cells and these are all linked together in one big list—the free list. As you create new lists, cells are taken from the free list and used. For example, if you typed (SETQ SHOPS '(tea milk sugar)) some cells would be taken from the free list and used. However, if you then changed your mind and typed (SETQ SHOPS '(biscuits socks)) LISP would use up more memory locations to store the new list. The problem is that the cells used to store the original list still exist linked together and have not been returned to the free list. Obviously, if this process continued, you would soon run out of memory.

Most LISP systems handle this problem by going on using up memory until the free list is empty. At this stage, a complicated program called the 'Garbage Collector' is brought into operation. This looks at all the cells in the computer's memory and discovers if they are still being used by the atoms and functions in the object list. Any cells no longer needed are then returned to the free list. After all the unused cells have been collected, normal processing resumes.

This can produce strange behaviour—your LISP program may be running and then suddenly stop for a few seconds, possibly in the middle of an important task, as the garbage collector does its work. Clearly, if the LISP program were in control of some highly interactive process, like the movement of a robot or flying plane, this would not be acceptable. In fact other methods of solving this problem will have to be developed for this type of situation. Some versions of BASIC use a similar garbage collection scheme to handle the storage of strings. Often there is a command with which you can force BASIC to do the garbage collection and some LISP systems have a function which does this.

MOVING ON

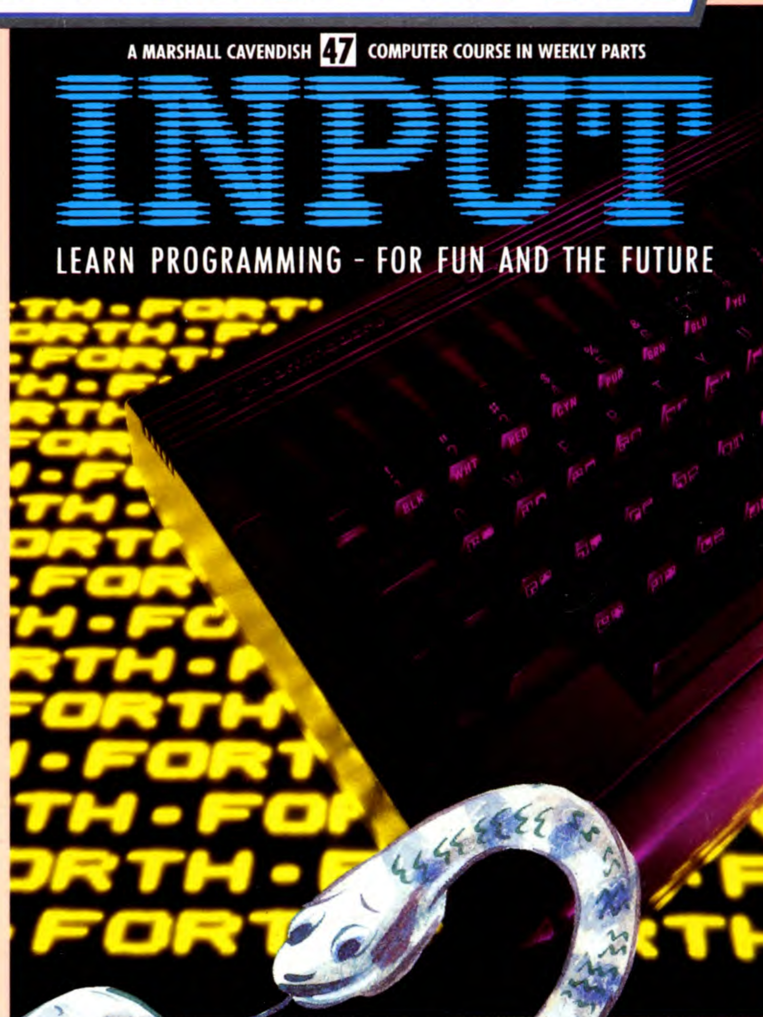
From the language of list processing to a language whose speed approaches that of machine code—the next language in the series is FORTH. Originally developed for scientific applications, these days FORTH is a powerful language with a wide range of applications.

CUMULATIVE INDEX

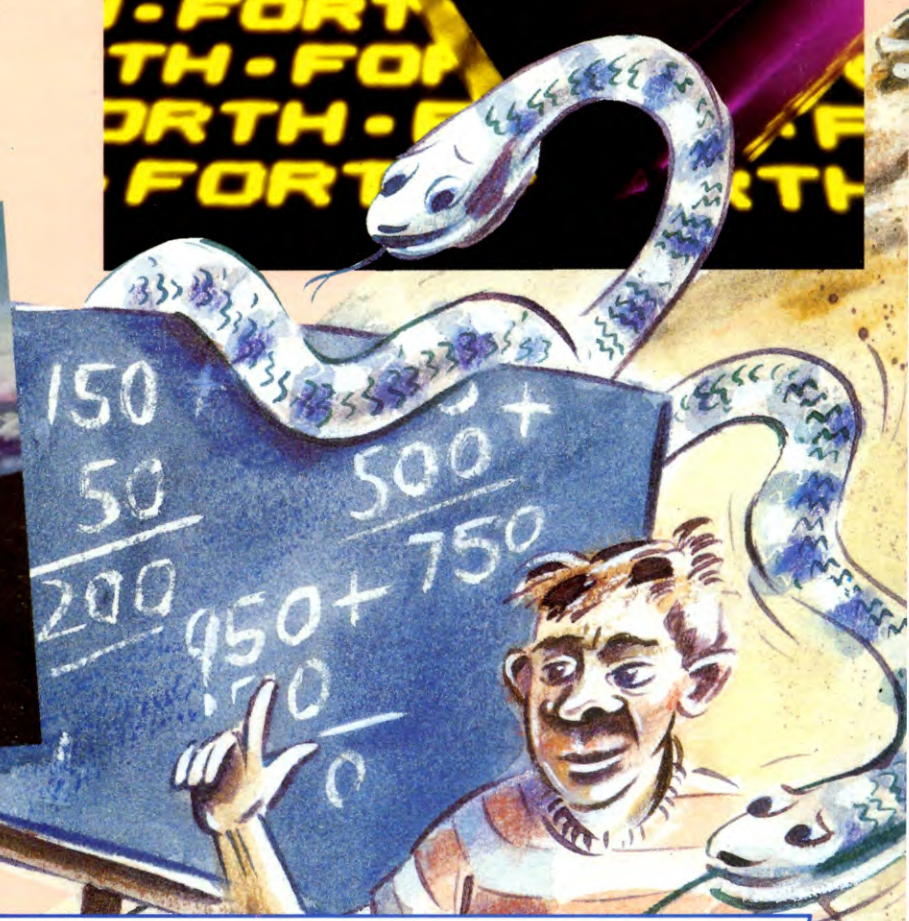
An interim index will be published each week. There will be a complete index in the last issue of *INPUT*.

The publishers accept no responsibility for unsolicited material sent for publication in INPUT. All tapes and written material should be accompanied by a stamped, self-addressed envelope.

COMING IN ISSUE 47...



- ***There are some ideas for giving depth to your GRAPHICS—GETTING THINGS IN PERSPECTIVE. Learn how to master vanishing points and foreshortening, and how to use shading to give solidity to your screen display.***
- ***Check out your TV or monitor with a simple TEST-CARD ROUTINE***
- ***Move on to looking at a new, high-speed language with the first of a three-part examination of FORTH***
- ***Complete your CRITICAL-PATH ANALYSIS PROGRAM and find out how to run it through a typical problem***
- ***In CLIFFHANGER, the next routine handles CALCULATING THE SCORE***
- ***Plus the next stage in ESCAPE, the BASIC adventure game program***



ASK YOUR NEWSAGENT FOR INPUT